HTML5 & CSS

Embedding Native Video

Introduction

Web video is evolving dramatically. I recently saw some figures that indicated that about half the data moved over the Web is video. Video files are large, of course, so that doesn't mean half of all Web activity is watching videos. But just about anywhere you go online, you'll find that video is part of the experience. One of the most radical and controversial changes associated with HTML5 is the emergence of native video, which plays in a browser without plug-in software. **With native video, you don't need Flash Player, Windows Media Player, or QuickTime Player.** Users no longer have to install and update different media players to watch video (or listen to audio). Instead, they can rely on browsers to seamlessly provide updated video and audio tools. Native videos look different in different browsers. Here's a native video looks like in Chrome:

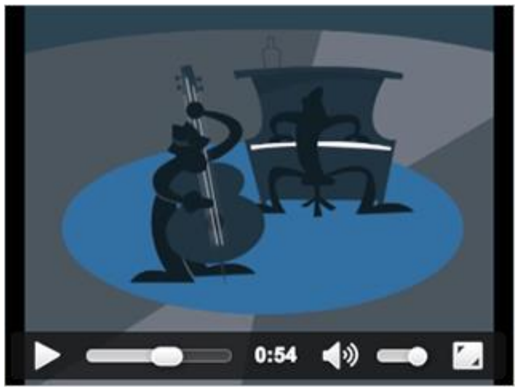


Figure 1 Watching native video in chrome

The same video displays with different controls in Firefox.

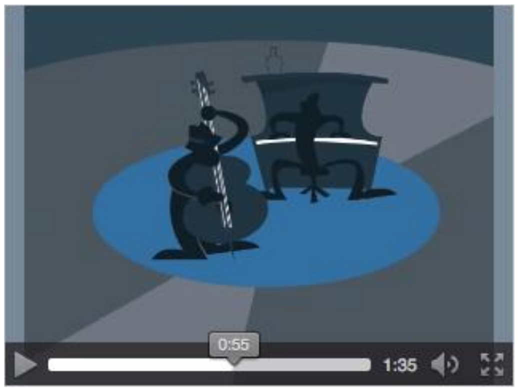


Figure 2 Watching native video in Firefox

No discussion of Web video can take place without acknowledging Flash Video. That technology made YouTube possible and launched the first wave of Web video. Flash Video was fast, high-quality, and played anywhere—as long as you installed the Flash Player.

This week, we'll explore what happened to the Flash Player plug-in and how HTML5 native video is replacing it. An overwhelming percentage of online video watching now takes place in environments that support HTML5 video. But not all HTML5 video is the same. Different browsers support different video formats, so we'll have to account for that as we embed native HTML5 video.

The Emergence of Native Video

Much of the impetus for the adoption and spread of native video came from Apple's decision not to support the Flash plug-in and Flash Video in its mobile devices. Apple's decision was the tipping point in the adoption of native video. But other developments created the foundation for that moment. It's worth reviewing the evolution of online video to understand where we are in the process and what it all means for advanced Web designers.

Phase one: All video required proprietary player software.

Windows Media files played in the Windows Media Player, and Apple's QuickTime video format played in the QuickTime Player. Windows users had a hard time accessing QuickTime video, and Mac users had difficulty playing Windows Media files. Also, both Windows Media and QuickTime files tended to be large, which discouraged distribution of longer or higher-quality video.

Phase two: Flash emerged.

Adobe's Flash Video (FLV) files required the Flash Player, but that plug-in software was available for any computer. The Flash Player looked and felt the same in any environment. Flash Video files downloaded faster than older formats because they used advanced compression software that reduced file size while maintaining quality. And the Flash Player did some of the processing that allowed video files to play, again reducing download size. On this basis, YouTube became a huge force on the Web, initially relying completely on the Flash Video format.

Phase three: Native video develops.

Other video formats began using the compression technology that made Flash Video files small and high-quality. And HTML5 included a new ***<video>*** element that browsers could support with their own, built-in, plug-in-free native video players.

Different Native Video for Different Browsers

Not all browsers support the same native video formats, and very old browsers don't support native video at all. What's the story here, and how do we address these factors when we embed native video?

Let's start with the different native video formats. You can find resources online that update you on the details of which browser supports which native video format, and I've provided information on those on CANVAS as reference. But the bottom line is this:

All current-generation browsers except Firefox and some versions of Chrome support the h.264 video format. These files have names that end with the letters MP4. Firefox and Chrome support the Ogg video format. In this format, the file usually ends with OGV. HTML5 supports a third video format: WebM. But no browser accepts only WebM video. We can provideHTML5 video that runs in any modern environment by providing just h.264 and Ogg options.

Hosted Video vs. YouTube

One option for providing access to video at your site is to embed video hosted at YouTube, Vimeo, or other video hosting services. This is an appropriate solution for beginning-level Web hosting. If you host your videos at YouTube, there's no coding involved. You simply copy and paste code from YouTube, and the YouTube video displays within your site. However, relying on YouTube or other video hosting sites has its problems: You don't control the video, the rights to the video, or how the hosting site presents it.

The hosting site harvests your visitor data and uses it for marketing. Your viewers will have the chance to see and perhaps be distracted by other videos. Those can include videos about or by a competing brand, band, resource, or product.

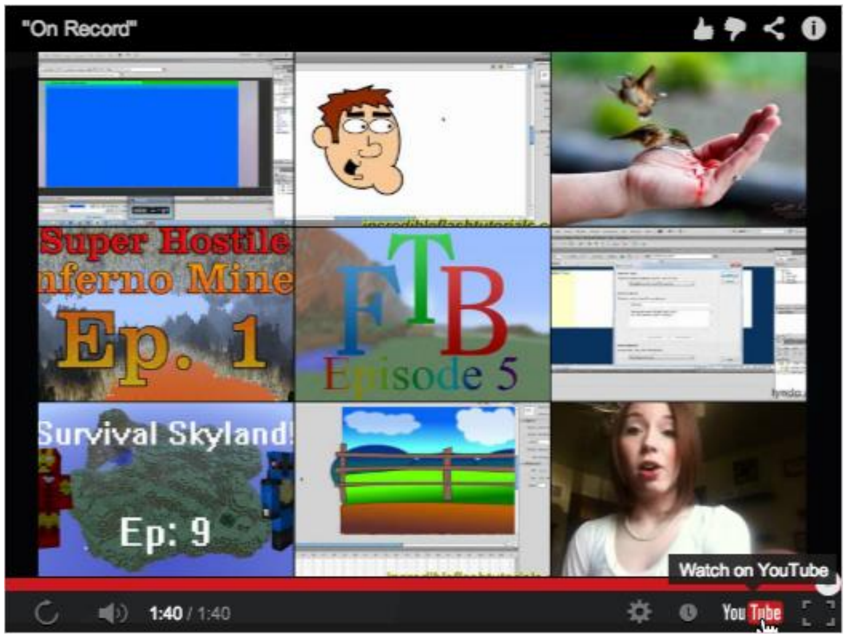
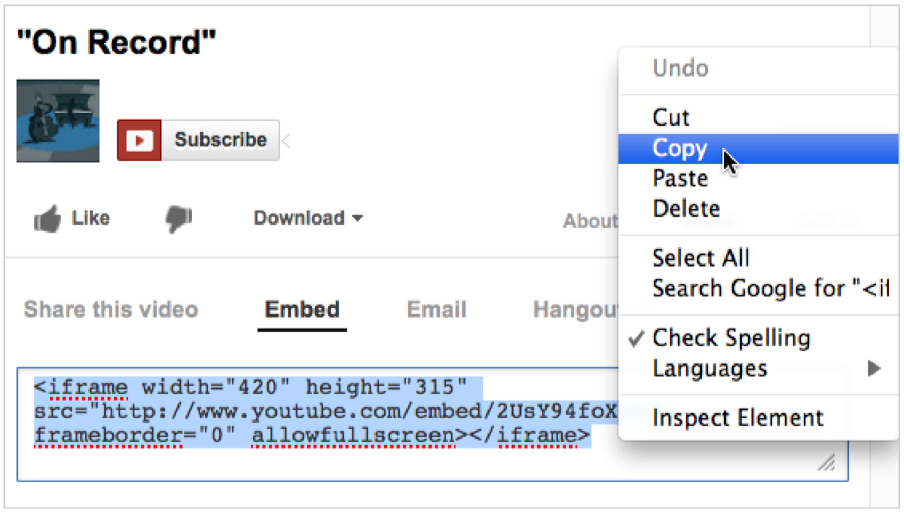


Figure 3 A YouTube video embedded in a Web page

For these reasons, advanced Web designers usually host their own video. But hosting services are useful in some circumstances. Every video hosting service allows you to generate an iFrame tag. This tag embeds content from another Web page within your page. You can think of it as a "window" into another page. The site from which you're embedding content determines the content of the iFrame element. But you define the size and location of the iFrame tag itself—the element that holds the embedded content.



```
<iframe width="420" height="315"
src="http://www.youtube.com/embed/2UsY94foX
frameborder="0" allowfullscreen></iframe>
```

Figure 4 Copying iFrame code to embed a YouTube video in a Web page

Where Does HTML5-Ready Video Come From?

If you capture video with a digital camera or recorder, it typically saves to a non-compressed format, like MOV (QuickTime) or WMV (Windows Media). If you use those formats, you need to convert them into HTML5-ready, compressed formats—Ogg or h.264. Since Ogg is a free, open-source project, you can convert video to Ogg easily. I'll provide a link on the CANVAS so that you can download the Theora Converter if you wish. (Theora is the name of the project that sponsors Ogg.)

On the other hand, the h.264 format is proprietary rather than open source. So, tools for generating h.264 video are generally associated with commercial video-editing software. You might have gotten such software free with your digital camera or video recorder, or it might be bundled with your computer. Professional video compression tools are part of many video editing programs. For instance, Adobe Premiere, Apple Final Cut Pro, iMovie, and the current version of Windows Movie Maker all export or convert video to h.264 format. You can download videos from Canvas. Save these videos in the folder you've been using for all the files for our class website.

Basic HTML5 Video Syntax

Let's go over the code you'll need if you want to embed a native HTML5 video:

```
<video>
<source src="filename.mp4" type="video/mp4">
<source src="filename.ogv" type="video/ogg">
</video>
```

This example defines different source files between the open and close **<video>** tags. It provides two options: an MP4 file that works with h.264, and an OGV file that works with Ogg. The example that I've just shown you is the type parameter. It tells browsers the mime type —that means the video file type and details about the compression technology. Most of the time, browsers can determine that information without a type parameter, but it doesn't hurt to include it. Generally, a type value of "video" followed by a slash and the abbreviations mp4 or Ogg is sufficient. If you're interested in fine-tuning video type specs, I've provided information on CANVAS.

Adding Specifications

The source parameter is the only required parameter for HTML5 video, and it may be all you need. It tells browsers what video file to present. But if you want to display controls (such as play, pause, and volume buttons and sliders), you'll need to add a parameter for that. You can also display videos at a set size using height and width parameters. Finally, you can define auto play, which starts a video playing as soon as a page opens. Please use this judiciously. Often users resent it when a loud audio track on a video begins playing when they open a page. They may be browsing in an environment where loud audio isn't appropriate, or they may just find it annoying to have an audio track begin playing without being told to. Let's walk through the syntax for all of these. If you want to use any of these parameters, add them to the open <video> tag. That way they apply to both versions of the video. To display controls, add this parameter: <video controls>

To define a width of 320 pixels and a height of 240 pixels (a common size for online videos), add this parameter:

<video width="320" height="240" controls>

And if you must add autoplay, use this parameter:

<video autoplay>

If you enable autoplay, you might consider adding the muted attribute that turns the audio track all the way down. Here's a video element with both autoplay and mute:

<video autoplay muted>

Safari, Firefox, Chrome, and Opera all support the muted parameter. Finally, you can loop a video to play repeatedly with this parameter:

<video loop>

Of course, you can combine parameters. Here's a video element with height, width, controls, and loop:

<video width="320" height="240" controls loop>

As with all tags, you'd separate parameters with spaces and put values in quotation marks. You don't need to include values for controls, muted, autoplay, or loop; you either include them or you don't. Current standard practice for non-HTML5 browsers is to include a line at the end of the <video> element (right before the </video> tag) with a message along the lines of: "This video requires HTML5, and your browser doesn't support that."

```
<video>
<source src="filename.mp4" type="video/mp4">
<source src="filename.ogv" type="video/ogg">
Your browser doesn't support the video tag.
</video>
```

Let's test all this out!

Embedding and Testing a Video

Download the videos from the CANVAS. We'll create a new HTML file to play the video. We'll use the same style.css style sheet we've been working from last week. We're going to be creating styles for the video, and whatever else is in your style sheet doesn't matter to us now.

With the style.css style sheet in place, open a new file in your code editor, and save it in your website folder (I've recommended that you create a folder called "class" on your desktop). Begin the file with this basic HTML:

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="UTF-8">
<title>Video Template</title>
<link rel="stylesheet" type="text/css" href="style.css">
</head>
<body>
<div id="wrapper">
<h1>Watch Our Video!<h1>
</div>
</body>
</html>
```

This basic page code defines an HTML5 document, has minimalist <head> content, and wraps all the page content inside our wrapper ID div. Now let's define the video element. If you feel confident, you can just put on your headphones and embed the video yourself. If you want some help, follow along with me:

1. Inside the wrapper ID, add the open and close video tags. I've highlighted them below.

```
<div id="wrapper">
<h1>Watch Our Video!<h1>
<video>
</video>
</div>
```

2. Add some basic parameters for the video. In this case, I'm going to display a controls bar and define a height and width for the video:

<video width="320" height="240" controls>

3. Add the source elements:

```
<video width="320" height="240" controls>
<source src="on_record.mp4" type="video/mp4">
<source src="on_record.ogv" type="video/ogg">
</video>
```

4. Provide fallback content for browsers that don't recognize the <video> tag. At a minimum, you should have a line of text explaining that users with noncompliant browsers can't see the video.

A friendlier and more helpful option would include some advice about what to do—for instance, you might link to another page in your site. Or you could provide a link to a browser that installs in any system and supports HTML5.

5. That's it! If possible, test your page in an Ogg-friendly browser (like Firefox, many versions of Chrome, or the mobile Android operating system) and an h.264-friendly browser (Safari).

Adding Posters, Styles, Preloads, and Controls

Sometimes the opening frame of a video isn't the most inviting way to display the video before a user clicks Play. One option for addressing this is to include a poster —an image file that displays in the video player when the video isn't playing. You can download image file from CANVAS, which I am using here. To apply this file as a poster, edit the video element as shown here:

<video width="320" height="240" controls poster="on_record.png">

Here's what you'll see.

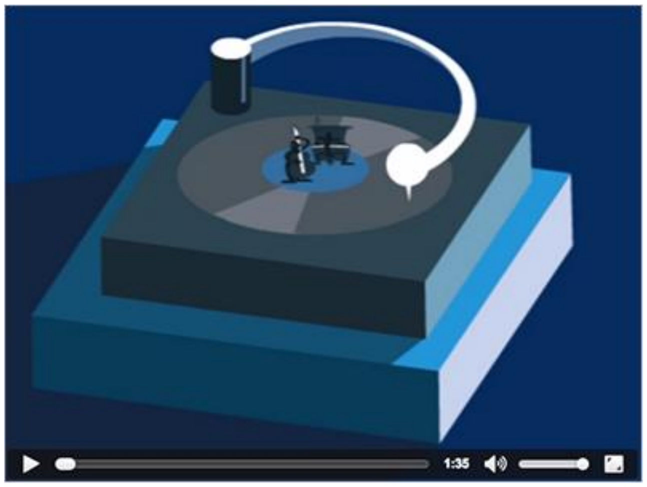


Figure 5 Testing a video poster

Styling the Video Element

One of the supposed benefits of native video is that browsers can define how video is presented. And, as we've seen, different browsers supply players that have different control icons. Well, that's nice for browser makers. It helps define the look and feel of their browser. But where does that leave Web designers who want to control how videos display? Plug-in players for QuickTime and Flash Video had options for designers to customize which controls displayed and how. The easiest way to customize the display of a video player is to define a style for the <video> element. Here's an example:

```
video {
    width: 400px;
    padding: 15px;
    margin: 15px;
    background-color:black;
    border:2px white solid;
    box-shadow: 10px 10px 5px black;
    }
```

The result looks something like this:

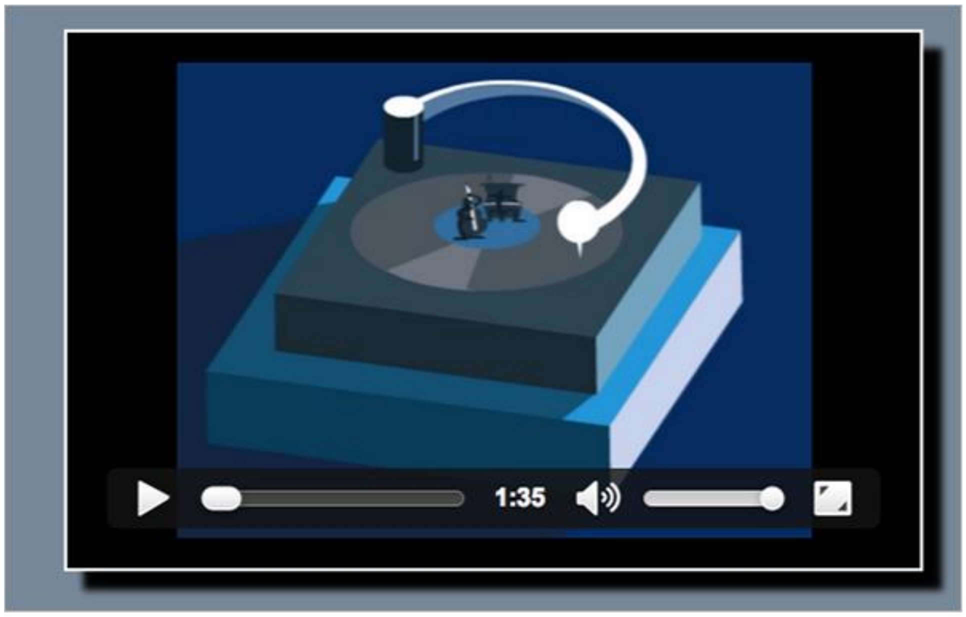


Figure 6 Applying CSS style to a video

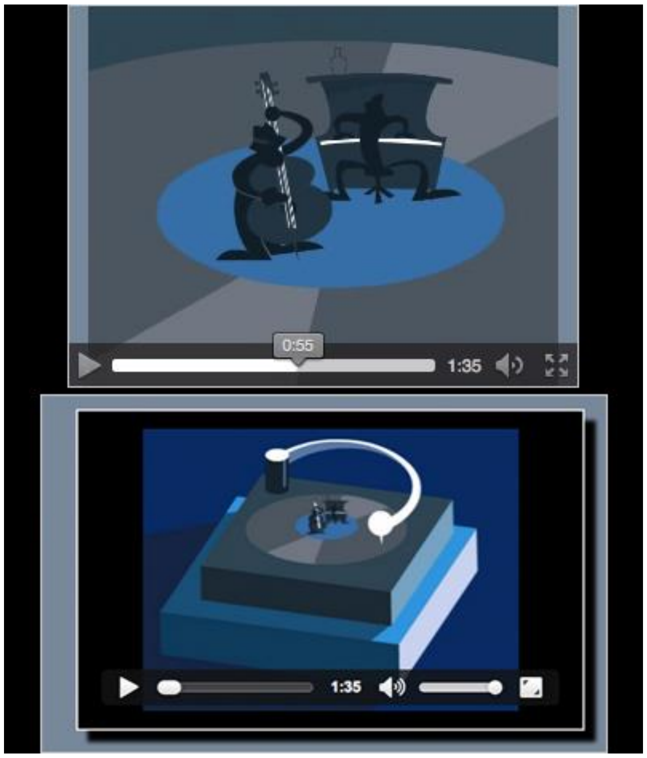


Figure 7 Difference between custom and standard video style

By customizing the display of the *<video>* tag, we've personalized how videos will look in our Webpages. We added a background that extends beyond the left and right edges of the video itself, we put a border around the video, and we added a box shadow. Even though the display of the video player itself is beyond our control, we've "framed" how it displays in our Web pages.

Preloading a Video

Video files are large. And it's a definite turn-off for visitors to click a video Play button and then twiddle their thumbs for 15 seconds while they wait. One solution is to add a preload parameter to the video tag that downloads a video when the page opens, even before a user plays the video. Here's an example:

<video width="320" height="240" controls poster="on_record.png" preload>

In some instances, auto-downloading a video slows down the opening of a page. A good rule of thumb is to use preload only when a video is the sole element or the main element on your page.

Using JavaScript Controls

We've seen that browsers define the look of native video players. And we've customized the look and feel of native video by defining a style for the *<video>* tag. The remaining option is using JavaScript to create control buttons. For deep understanding of JavaScript code wait for couple of weeks. Here we'll experiment with a simple example—creating a play/pause button. To do that, add this code after the close *</video>* tag:

```
<br>
<button onclick="playPause()">Play or pause the video</button>
<script>
var myVideo=document.getElementById("video");
function playPause()
{
if (myVideo.paused)
myVideo.play();
else myVideo.pause();
}
</script>
```

The first line of this code creates a line break <br> tag. The second line creates the play/pause button and the JavaScript within the *<script>* element defines the action associated with the button. JavaScript almost always links to HTML through defined ID style selector. (We explored ID and class selectors in Week-1.) In this case, we've defined our JavaScript snippet so that it applies to an ID named "video." To connect this script with our video, we need to add the video ID to our <video> tag. Here's how to do that:

<video controls poster="on_record.png" id="video">

Here's how the video looks now. You can see the JavaScript play/pause button at the bottom.

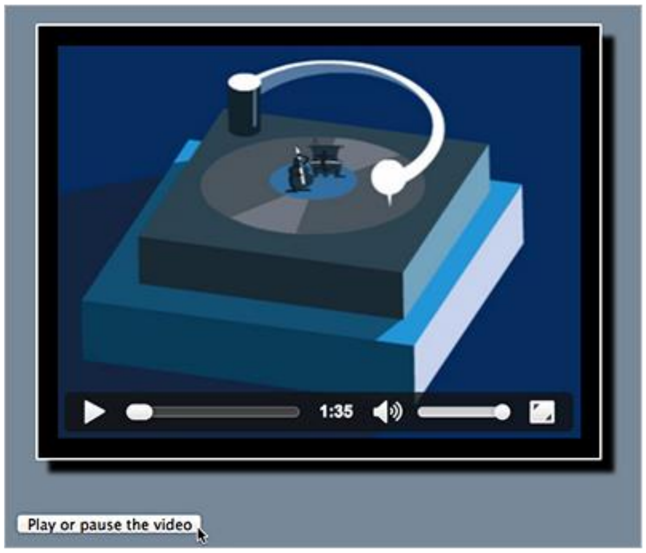


Figure 8 Testing JavaScript Play/Pause Button

It's easy to customize the text in the button; simply replace the "Play or pause the video" text inside the <button> element with other text. We can customize the button style as well. For example, we might define a button style like this:

```
button {
    background-color:black;
    color:white;
    height:40px;
    border:2px white solid;
    box-shadow: 10px 10px 5px black;
    border-radius: 5px;
    }
```

That would give us a button like this:

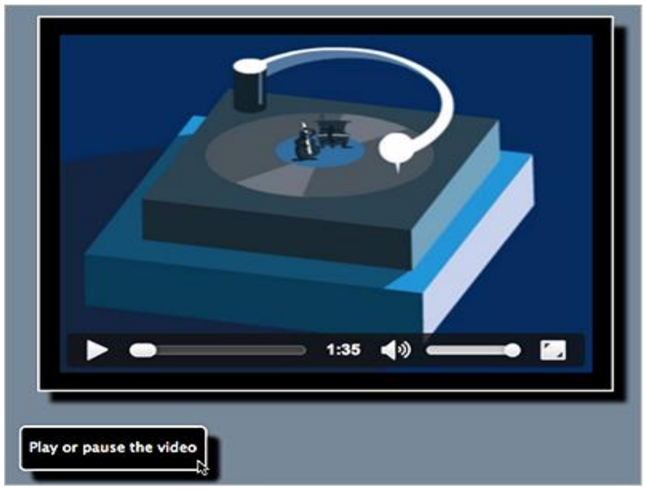


Figure 9 Defining Play/Pause Button

And we might even add a hover state to the button to provide a bit of animation, like this:

```
button:hover {
    background-color:beige;
    color:black;
    height:40px;
    border:2px white solid;
    box-shadow: -10px -10px 5px black;
    border-radius: 5px;
    }
```

My example looks something like this when I hover my mouse over it. See the difference?

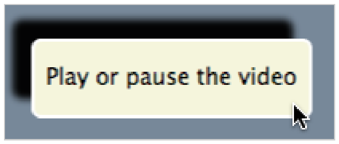


Figure 10 Testing a hover state for a video play/pause button

How will you use this native video approach in your assignments?

Now You Try

I'll let you customize the colors for the button. To reinforce your sense of color scheming, apply the color scheme that you defined in Week-1.

Why HTML5 Video Is the Future

How does HTML5 video fit into the wider realm of Web video? First of all, yes, there is a wider realm of Web video. Most desktop browsing environments support Flash Player, but most sites that cater to "coveted demographics" (as they say in advertising) don't support the Flash Player plug-in. Since iPhones and iPads don't support Flash Player, and since iPad and iPhone users are disproportionately active online and buy disproportionately more stuff online, most Web designers avoid plug-ins that aren't accessible in the iOS (Apple mobile) operating system. Also, proprietary video formats that require a plug-in are still in wide use. Not every Web designer is going to immediately pull all of his or her videos from a site to comply with HTML5 specs. Besides Flash video (FLV), proprietary video formats still in use include: QuickTime video (MOV), Windows video (WMV, AVI) and Real Media (RM) files but HTML5 native video has rapidly eclipsed older formats.

What are HTML5's advantages? For designers, HTML5 video is fairly easy for us to code. It includes such features as posters, optional control display, optional autoplay, optional preloading, and optional muting. For users, it provides a seamless experience that reflects the look and feel of their browsing environment. The main challenges with HTML5 native video are:

- No one file format is supported by all browsers.
- Designers can't do much to customize the player controls.

How do we address the challenges?

- Provide Ogg and h.264 options for every video we embed.
- Rely on CSS styling to customize the look and feel of our videos.
- Consider using a JavaScript play/pause button to add customization to our video interface.

What About Mobile Users?

HTML5 plays just fine in mobile sites. All mobile browsers support HTML5, so you can play HTML5 video in those environments. You don't have any more freedom to customize HTML5 video in mobile sites than you do in full-sized sites, including sites created with jQuery Mobile. But you can safely use HTML5 video in any mobile-friendly site.

Building Forms

Introduction

Now we're going to explore one of the most powerful and underrated elements you can add to a Web page: the form.

Forms are everywhere on the Web—and for good reason. They're the most effective and versatile way of getting information from your website's visitors, and that makes them an essential part of Web design. Forms can be simple. Perhaps the single most popular type of form on the Web is the search box. The minimalist page design at Craigslist, for example, centers on a search box.

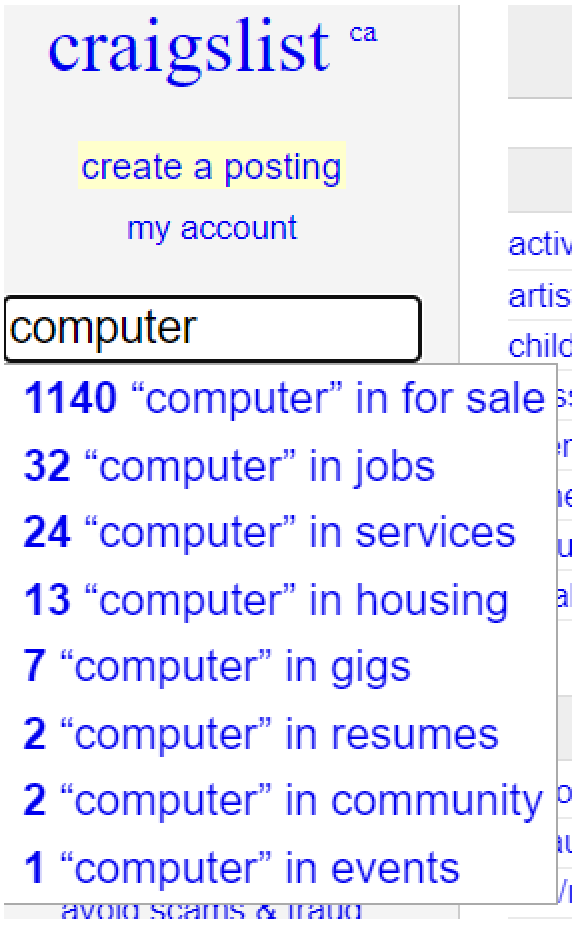
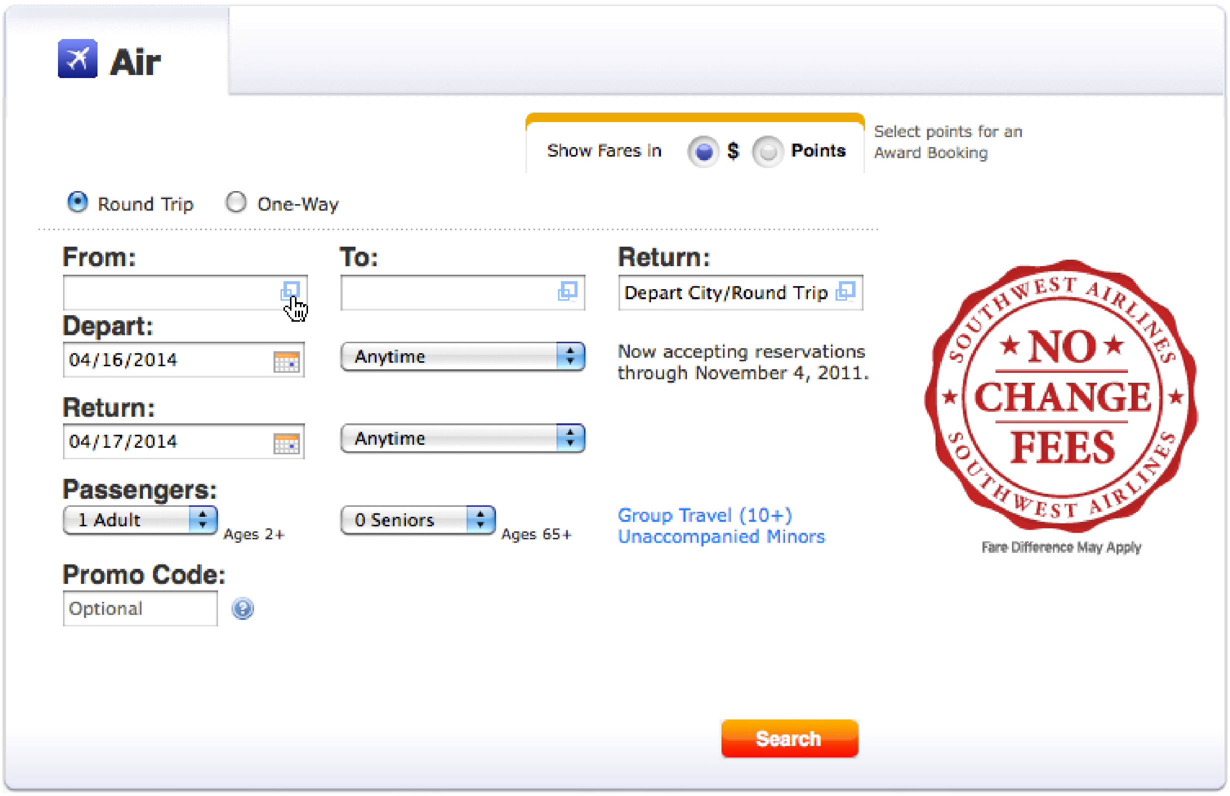


Figure 11 The search box at Craigslist

Other forms are complex. For instance, I count at least 13 fields in the form below for booking flights at Southwest Airlines. They range from a field for choosing to view fares in dollars or points to a field for entering a promotional code.



Air

Show Fares in [✓ $] Points

Select points for an Award Booking

[✓ Round Trip] One-Way

From:	To:	Return: Depart City/Round Trip
Depart: 04/16/2014	Anytime	Now accepting reservations through November 4, 2011.
Return: 04/17/2014	Anytime	
Passengers: 1 Adult Ages 2+	0 Seniors Ages 65+	Group Travel (10+) Unaccompanied Minors
Promo Code: Optional		

Search

Figure 12 Exploring flight options at Southwest Airlines

Adding a working form to your site involves four steps:

1. Create a form in HTML.
2. Add form fields.
3. Make the form accessible and inviting.
4. Define where the form data will go and how it'll get there.

The Role of Forms

Forms are everywhere. Just about every substantial website uses them. Without forms, people couldn't buy things online, participate in online polls or forums, post to Facebook, or sign up for an online newsletter. They couldn't search with Google or Bing or book a flight at an airline's site. Some of these examples involve complex forms. They require a back-end database that collects, sorts, organizes, and responds to form input instantly. When you buy a book at Amazon, for example, form data goes into computers that calculate shipping and sales tax, check and update inventory stocks, and later generate that annoying suggestion that since you ordered a book on back pain, you might be interested in a new book on aging gracefully. If you're interested in building these kinds of back-end scripts and databases, wait for few weeks and we will be there. Other forms, however, don't require such sophisticated databases. A feedback form, for example, is a way to get free advice and input from visitors to your site. A visitor simply fills out a form that automatically goes into his or her email client (such as Microsoft Outlook, Windows Live Mail, Apple Mail, or freeware Eudora) and makes its way to you. You can even collect data that will enable you to build mailing lists and feedback forms for activist groups, fan clubs, support groups, and so on.

Server-Side and Client-Side Form Actions

Let's get technical for a moment. Obviously, form data has to go somewhere, or what's the point? Somehow, the information a visitor enters into a form has to lead to some kind of action. This can happen in two ways:

1. Client-side form handling
2. Server-side form handling

Client-side form handling means that the management of a form's input happens entirely within a browser (client is essentially another term for browser). The most typical example of a client-side form is a jump menu for navigation. When a visitor chooses a page from a jump menu, a code that's part of or linked to the Web page itself manages that input and sends that visitor to the page they choose.

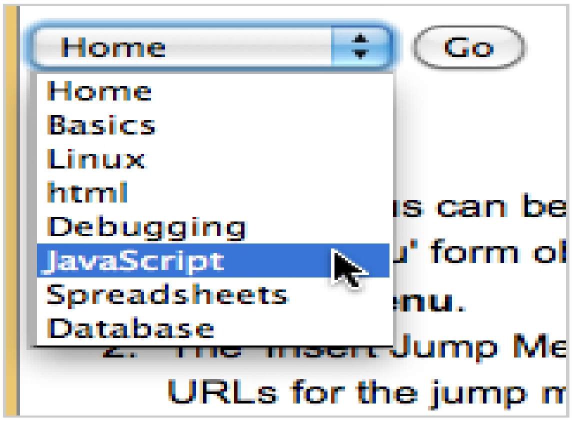


Figure 13 A client-side jump menu form to navigate a tech-support site

You probably already know that a server is a huge, centralized computer that manages vast amounts of data. Server-side form handling sends form data to a server, which stores it. Sometimes other scripts running on the server act on that data. Even sending form data by email is server-side form handling, since the data goes over an email server.

What You Can Do with Forms

As a Web designer, you may be involved with forms in three ways.

1. You may find yourself in a large-scale Web development environment where you work closely with back-end programmers. In this case, you'll design forms, and programmers will develop code on servers to process the information that users submit.
2. Or you may be a one-person-team creating an entire website. In that case, you can define simple form-handling processes using email, as we'll explore this week.
3. You might want to use online resources that provide server hosting and scripting support. Many of these resources offer free scripts and hosting. For instance, some will manage mail lists of less than 500 names. They allow you to collect names and email addresses through forms linked to their server scripts and then generate mailings to that list. We'll explore those later.

In all these scenarios, Web designers build forms that are easy to use, attractive, and reliable. So, let's get started doing that right now!

Working with Signup and Feedback Forms

One of the most useful and easy-to-make forms is a signup form. It collects the names of people who want to hear from you, and I've already talked about how valuable that is. We've also talked about feedback forms. Having one of those on your site accomplishes several things:

1. It shows you care what people think of your site.
2. It allows you to enlist visitors as free proof readers and testers who report errors or glitches in your site.
3. It can be a doorway to a signup form, where visitors who share feedback feel they're part of a community and can stay in touch.

To create a feedback form, open a new file in your HTML code editor, and save it as form.html. Enter or paste this code into the form.html file. This code defines a document title and includes some introductory text that encourages users to fill out the form:

```html
<!DOCTYPE HTML>
<html>
<head>
<meta charset="UTF-8">
<title>Form</title>
<link href="style.css" rel="stylesheet" type="text/css">
</head>
<body>
<div id="wrapper">
<h1>Feedback . . . Stay in Touch</h1>
</div>
</body>
</html>
```

This HTML file has a link to a style sheet called style.css, so we can use the same style sheet we've been using for other pages in our site.

Defining a Form

Now that our basic page is set up, it's time to add the form. We'll use the form tag to define the form. Remember, the form tag encloses everything that's part of the form. So, as you add form fields to collect information such as name, email address, rating, and comments, be sure to place those form fields inside the <form> and </form> tags. We need to define several properties in our form as we set it up:

Action: The form action defines what happens to the form when a visitor submits it. While some forms connect to complex databases that database experts manage, we'll handle our form more simply. When a visitor clicks Submit after filling out the form, their email client will launch, and the content of the email will consist of the data the visitor has entered into the form.

Method: The form method determines how to submit the form. There are two form methods: **get and post**. In most cases, you'd use a get form method when downloading data into a form from a server and a post form method when sending data somewhere.

Enctype: The form enctype (which stands for encryption type) defines how to handle the text in the form. If you were posting data to an online database, you'd probably use "multipart/form-data." But we're going to use a simpler enctype, "plain text," that uploads data as regular text.

Name: Every form has a name. We've named ours form.

Let's define a form that submits the entered data to an email address. Enter the following code after the close of the </h1> tag. (If you want to test the form by linking it to your email address, substitute your email address for the placeholder content.)

```
<form action="mailto:email@email.com" method="post" enctype="text/plain"
name="form">
</form>
```

This is a mail-to action. In real life, you'll replace "email@email.com" with an actual email address you setup to get feedback. And you can define subject line text by adding *?subject=xxx* to the form action, as in this example:

```
<form action="mailto:email@email.com" ?subject=Signup Form method="post"
enctype="text/plain" name="form">
</form>
```

There's nothing to preview yet, as the form itself doesn't show up in a browser. But we've created the basic content packager for a form, and we're ready to add form fields.

Creating Form Fields

Form fields are different ways of collecting data in a form. Text boxes (which hold a single line of text) and text area boxes (which can hold multiple lines of text) allow visitors to enter email addresses, names, phone numbers, credit-card numbers, and comments. Other kinds of form fields present people with choices. A select menu, for instance, provides a list of choices. This is a convenient way of letting people choose a topic in a search form—for example, the person's job title or the person's preferred date to book a flight.

Check boxes and groups of radio buttons also present choices. Visitors can use a set of radio buttons to rate a site on a scale of 1 to 10 or to specify the type of credit card they're using. And check boxes allow someone to say yes or no to an offer or to sign up for a newsletter. Let's see how you can populate your form.

Adding a Text Field

The most basic, most widely used, and most flexible way to collect data in a form is through a text field. To define a text field, use the HTML *<input>* element. The rules for naming form fields are similar to those for naming files: **Avoid special characters (such as ? / * @) and spaces**. You don't have to restrict names to lowercase. If you wish, you can define the size of the field in characters. The syntax for a text field called name (which you might use to collect a visitor's name) that displays 32 characters is:

```
<p>
<input type="text" name="name" size="32">
</p>
```

Add this code inside your <form> element! Remember, all form fields have to be inside the <form> and </form> tags. So, if I zoom out a bit, the code for the entire form now looks like this:

```
<form action=mailto:email@email.com ?subject=Signup Form method="post"
enctype="text/plain" name="form">
<p>
<input type="text" name="name" size="32">
</p>
</form>
```

The most accessible way to tell visitors what information to enter into a form field is through a label element. These usually go before a form field. Labels are associated with input field IDs. Every label uses the syntax `<label for="ID">Name</label>`, where the value of "name" is the same as the name parameter in an input field. That's more complicated to explain than it is to demonstrate, so here's our form with a label added to the name field:

```
<form action=mailto:email@email.com ?subject=Signup Form method="post"
enctype="text/plain" name="form">
<p>
<label for="name">Name</label>
<input type="text" name="name" size="32">
</p>
</form>
```

Use the following code to add fields that collect name and email information (setting the size of the mail field at 40 characters), along with heading-3 text telling visitors what to enter into the form. Place this code above the closing form tag.

```
<p>
<label for="email">Email</label>
<input name="email" type="email" size="40">
</p>
```

“By using an email input type, we are telling browsers to check to see that whatever is entered here looks like an email address. And we are telling mobile browsers to provide a keypad for users that facilitates entering an email address.”

You can save your HTML file and open it in a browser to see the two form fields you've created so far. The form isn't "submittable" yet—we'll need to add a submit button for that. But you can enter a little information into the fields to get a feel for how visitors will experience them.

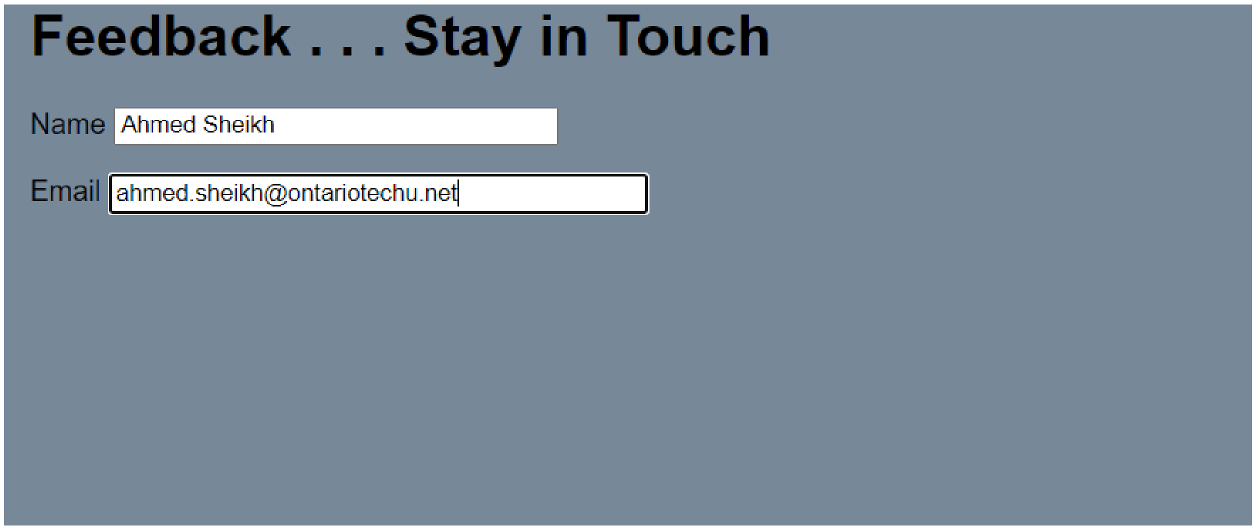


Figure 14 Testing the text fields

Adding a Text Area Field

Because text area fields allow multiple lines of data entry, they're great for collecting comments. The syntax for a text area field requires a name, as all fields do. But instead of a size, you define columns (the number of characters someone can type into a line) and rows (the number of lines). Add the following code above the closing </form> tag to define a form field called comments that's 60 characters wide and six rows high. The code includes a heading that lets visitors know what to enter in to the field.

```
<p>
   <label for="comments">Comments?</label><br>
   <textarea name="comments" cols="60" rows="6"></textarea>
</p>
```

Here's another opportunity to save and preview the form. Put yourself in the shoes of a visitor to your site, and enter a comment.

Note

Can I Set Limits on the Length of User Messages? HTML5 has a new parameter to define a maximum number of characters in a textarea field. The syntax is maxlength="x" where "x" is a value. This only applies in browsers that support HTML5.

Defining Choices

Radio button groups are similar to select menus, which pop up with a set of predefined choices. I like select menus better than radio button groups, but both provide visitors with a set of choices. As we'll see later, select menus are much more accessible in mobile devices than tiny radio buttons are.

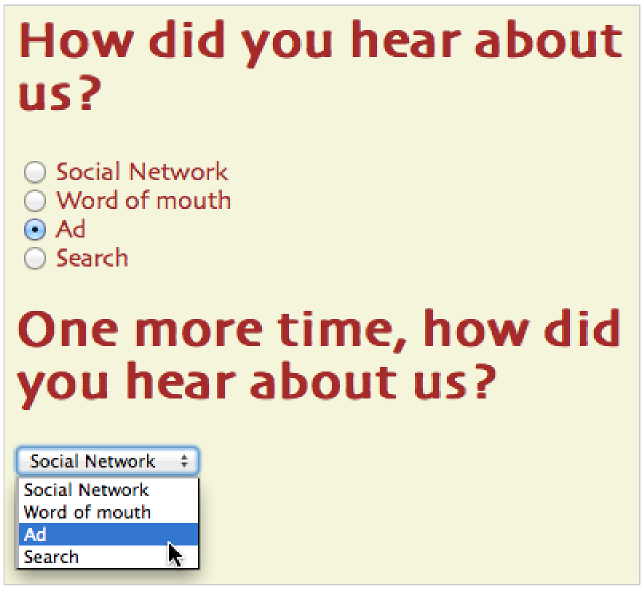


Figure 15 Radio button and Select Menu

Check boxes provide simple choices. Let's add one to our form.

Adding a Check Box

Check boxes allow visitors to opt in or out, choose yes or no, or make some other choice that has two options. It's an ideal type of field to allow people who fill out your feedback form to get on your email list (or not). Check boxes require just one property: a name. Add the following line of code before the closing </form> tag:

```
<p>
   <input type="checkbox" name="maillist">
   <label for "maillist">Put me on your e-list!</label>
</p>
```

Note that the label explaining what the check box does ("Put me on your email list!") comes after the checkbox. Check boxes and radio buttons are the only kinds of form fields where people expect to see an explanation of the form field after they see the form field itself.

You can define a check box as selected by default by adding the parameter "checked" inside the check box element. For example:

<input type="checkbox" name="maillist" checked>

Is that ethical? Is it appropriate? That depends on circumstances, and it's your call.

Adding a Select Menu Field

Select menus are a good way to provide visitors with a set of choices. You'll often see them on forms that ask what state or province you're in. Select menus have names and a set of options. Here's the syntax for an option menu named Where. It lets visitors tell you how they found out about your site. The code includes text to explain the form field. Add this code to your file—put it just before the closing </form> tag:

```
<p>
        <label for "where">How did you find us?</label>
        <select name="where">
        <option value="Search engine">Search engine</option>
        <option value="Word of mouth">Word of mouth</option>
        <option value="Craigslist">Craigslist</option>
        <option value="Other">Other</option>
        </select>
</p>
```

Of course, you can change the value and the display text for any option. And you can add as many options as you like. (I often run into lists of 50 when choosing "State" on many forms.). Let's see how the form looks when you save your page and preview in a browser. If you're following along step by step, your form should look something like this:

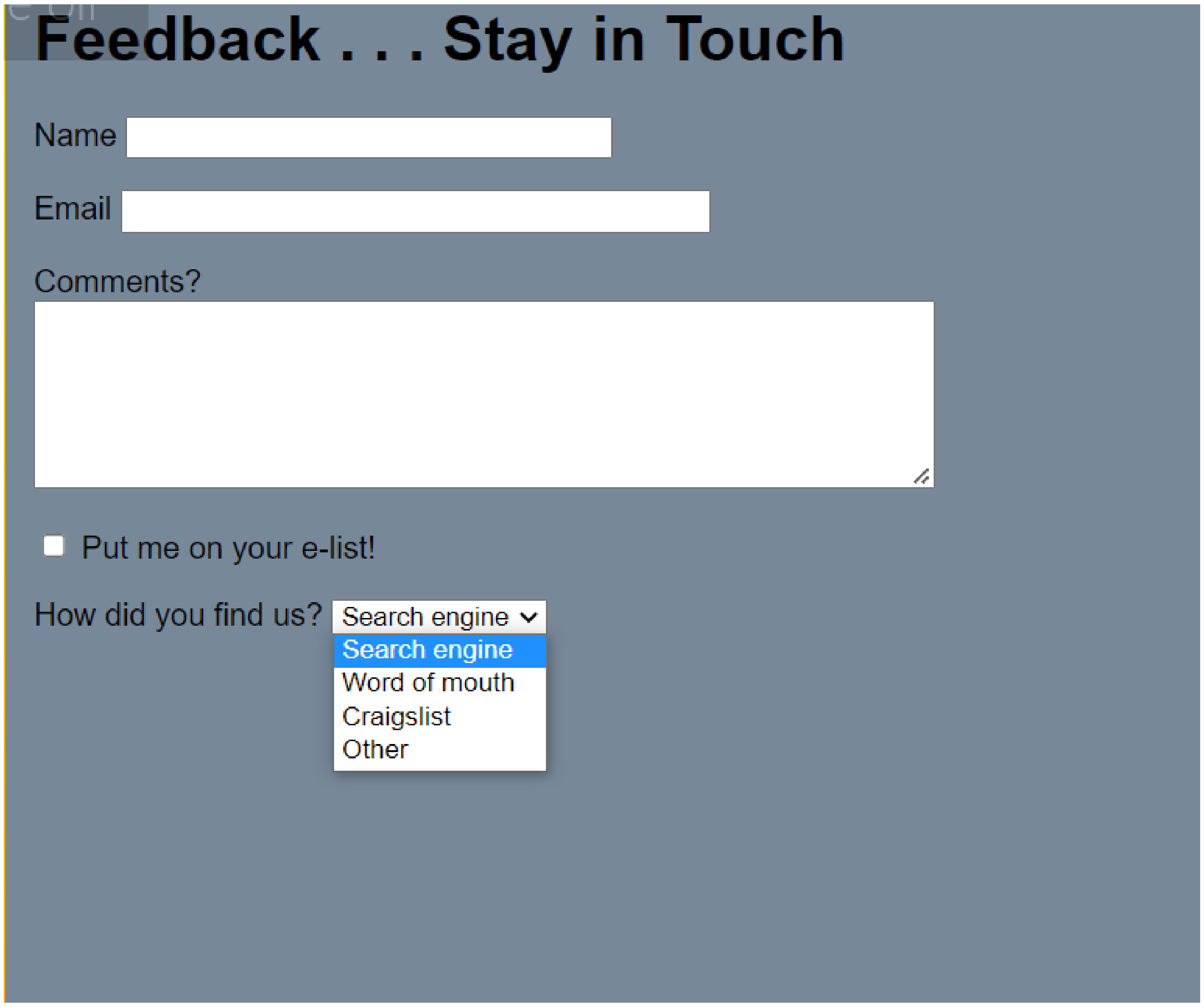


Figure 16 Testing check box, text area, and option menu

Finishing the Form

Now that you've created a variety of form fields, it's time to add buttons that allow visitors to submit the form or to reset it (clearing all the fields) if they aren't happy with the content they entered. We'll also review and test the action property we created to submit the form through the email program on a visitor's computer.

Adding a Submit Button

The submit button triggers the action that handles the form data—in this case, launching an email program with the form data that's already in the email. The syntax for a Submit button is:

<input type="submit" value="xxx"/>

Where you see xxx, substitute the text of the label that will appear on the form button. Add this code to your form just before the closing form tag. Remember: All form content must be with in the <form> and </form> tags!

```
<p><input type="submit" value="Click here to submit the form"/></p>
```

We'll test that button when we review and test the action.

Adding a Reset Button

Reset buttons are optional, but they're a helpful service for people who want to start over when filling out a form. The syntax for a reset button is similar to that of a submit button:

<input type="reset" value="xxx"/>

Instead of xxx, type in the text of the label that will appear on the submit button. Add this code to your form just before the closing </form> tag. Again, all form content must be within the <form> and </form> tags, or the button won't work.

```
<p><input type="reset" value="Click here to reset the form"/></p>
```

Test the reset button by saving the HTML file in your text editor and opening it in a browser (refreshing or reloading the browser window if necessary). The reset button should clear all data you entered into your form.

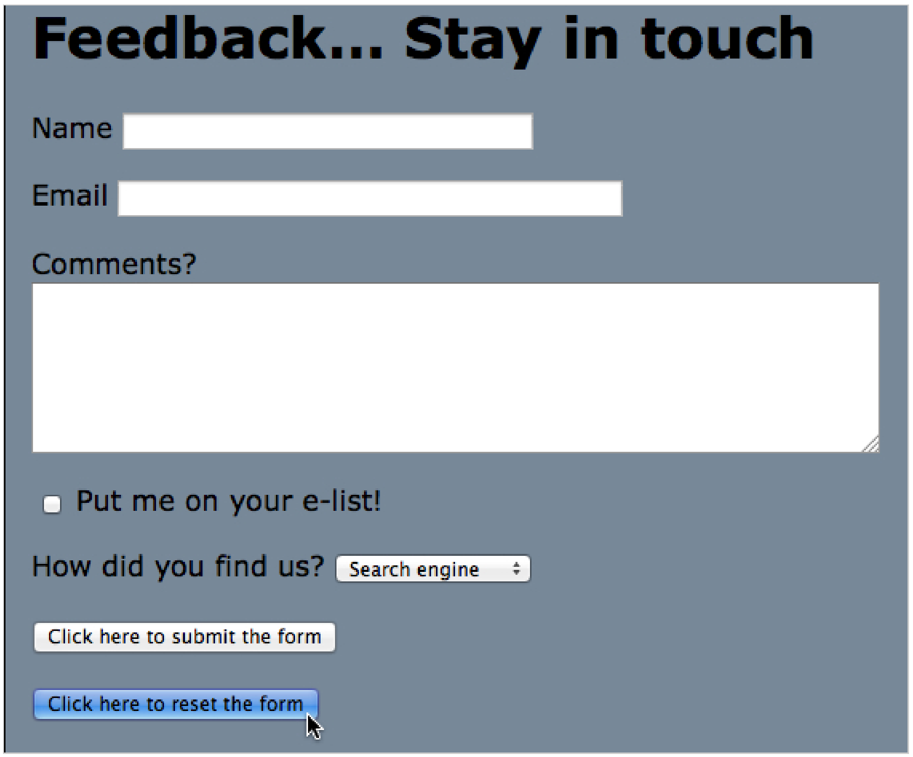

Feedback... Stay in touch
Name
Email
Comments?
Put me on your e-list!
How did you find us?
Search engine
Click here to submit the form
Click here to reset the form

Figure 17 Testing a reset button

Testing the Submit Button

The submit button in our form activates the form action property. In a professional-level form that connects to some kind of script running on a server, the form action links to a script on a server. Often that script is in the PHP scripting language. Other commonly used back-end scripts include Ruby, Perl, ASP, and JSP. We'll generate a back-end script to manage form data. For now, our simple email action will be enough to test the form.

Submitting form data through an email won't work in some configurations of Windows email clients. Our form action relies on the browser launching an email program (such as Apple Mail, Microsoft Outlook, Windows Live Mail, or Eudora) when a user clicks on an email address in his or her browser. However, some Windows environments lack the configuration to launch the defined email client when a user clicks the email link. If that's the case, then our testing scenario won't work.

Enhancing Forms and Collecting Form Data

Forms are arguably the single most valuable element of a website. They let you sell products and accumulate e-lists, and they let visitors search your site, order products, and do much more. In many ways, forms are the way you harvest content through your website. So, forms should be inviting and accessible . . . even fun! And yet, I'll bet if you surveyed a thousand people who spend a lot of time online, not many of them would list "filling out forms" as their favorite online activity. Let's look into the new features of HTML5 that make it easier, less stressful, and even fun to fill out forms. Those tools include:

- Placeholder text that prompts a user on what to enter into a field

- Datalists to save users time in typing entries
- Output fields that make calculations for a user
- Validation rules to help users enter data that makes sense

Now, I'll show you how to collect form data in a more professional way than the mail-to action we just did. Let's get started by making it easier for your visitors to fill out forms.

Making It Easy to Fill Out Forms

The easier we make it to fill out a form, the more likely we are to entice visitors to use that form. HMTL5 provides two valuable tools that make forms less hassle:

- Placeholder text
- Datalists

Placeholder text provides clues as to what a user should enter into a form field. Datalists use what programmers often call "autocomplete." If a visitor begins to type something into a form that beings with the letter **a**, for instance, he or she can see a list of options that begin with that letter. We'll return to these two techniques in just a moment, but first let's get our form set up.

Form Basics: A Quick Review

To keep things focused, let's create a basic form to test the new features in HTML5. Create a new document in your code editor, and save it as form2.html. Use this code for the entire page, including the form:

Defining Placeholder Text

You can use placeholder text to supplement a form label, like this:

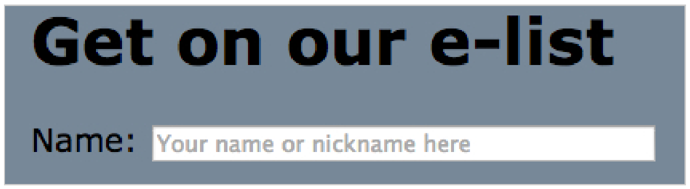


Figure 18 A form field supplemented with an HTML5 placeholder text

Or you can use placeholder text instead of a form label, like this:

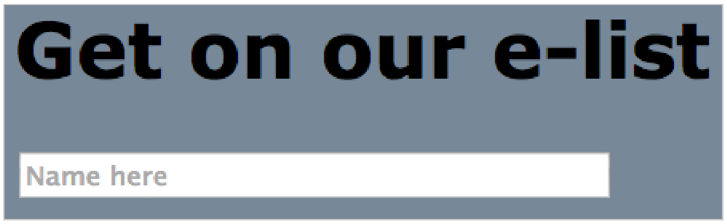


Figure 19 A form field that relies on HTML5 placeholder text

The second technique is appropriate for pages that people will access only with modern browsers. Here's the syntax for a form field with placeholder text, where fieldname is the input type (like text or email).

```
<input type="text" name="fieldname" placeholder="placeholder text">
```

And here's an example of how we can add to a placeholder text field to our form now:

```
<p><label for="name">Name:</label>
<input type="text" name="name" placeholder="Your name or nickname here"
size="48"></p>
```

Note that I included a few basic form techniques we discussed in the last lesson:

- The <label> element provides information for users who lack support for HTML5 placeholders.
- There's now a "size" property that allows the input field to display a longer field.
- We're using a paragraph (<p>) element to style the label and text field.

Defining a Datalists

Datalists are similar to the select menus, but datalists can filter results. So, if you ask a user to choose his or her country of origin from a list, you don't have to display a list of 200 or so countries. As soon as a user types an A, a list appears restricted to countries that begin with A. And when a user types a second letter, like L for example, the list shrinks to Albania and Algeria. This is more inviting and less cluttered than displaying long lists of options.

Another difference between a select menu and a datalist is that with a datalist, users can enter content that isn't on the list. That has advantages and disadvantages. If you want to constrain users to certain choices, stick with a select menu. But if you just want to make data entry less of a hassle for users and help those of us who are spelling-challenged, then a datalist is a good way to go.

Here's the syntax for a datalist:

```
<input list="list">
<datalist id="list">
<option value="first option">
<option value="second option">
<option value="third option">
</datalist>
```

And here's an example we can use:

```
<p><label for="countries">Where do you live?</label>
<input list="countries">
<datalist id="countries">
    <option value="Afghanistan">
    <option value="Albania">
    <option value="Algeria">
    <option value="Andorra">
    <option value="Angola">
    <option value="Antigua and Barbuda">
    <option value="Argentina">
    <option value="Armenia">
    <option value="Aruba">
    <option value="Australia">
    <option value="Austria">
    <option value="Azerbaijan">
</datalist>
```

Okay, I cheated on that list. In real life we would have included all countries. But this is a class assignment, and I didn't want to use up all our time and space with the full list. But this abbreviated list of countries is enough to experience the concept. When someone begins to type a state name, that person sees a prompt with a filtered list of options, like these:

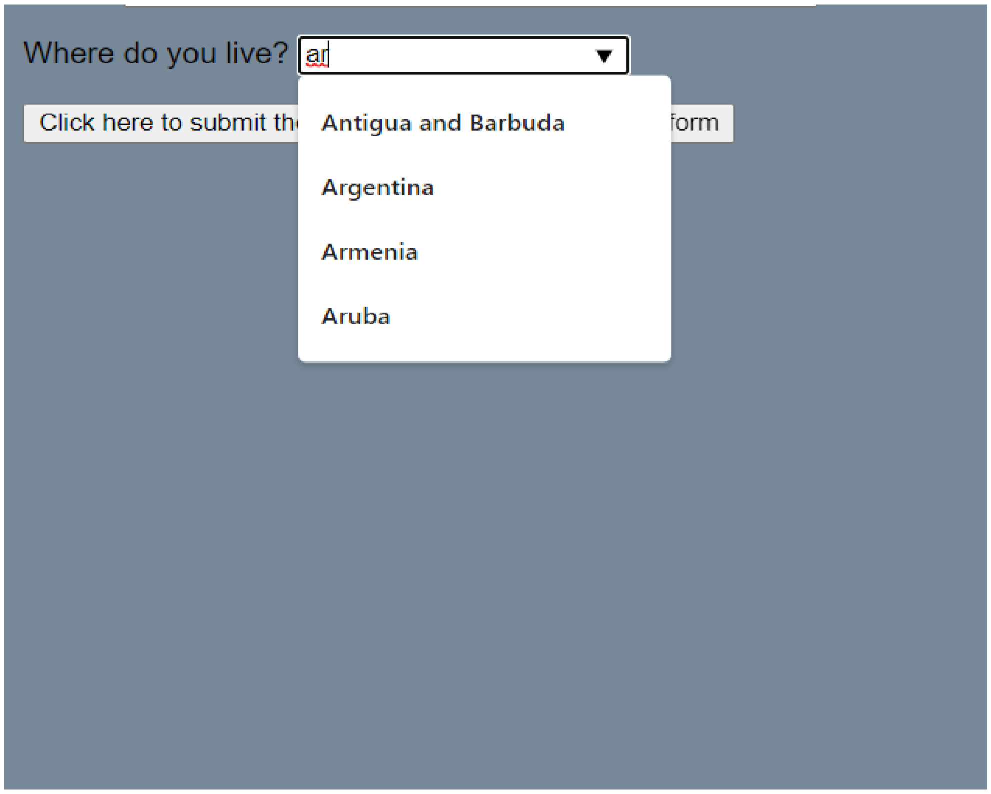


Figure 20 Using an HTML5 datalist to make data entry easier

Now let's talk about another way to make it easy for your users to fill out your forms.

Using Form Output Elements

HTML5 form output elements perform calculations for a user. You can use them to create complex forms that more or less replicate calculators. I'll provide links to resources for that kind of application on CANVAS for reading. The basic syntax for this kind of output element is:

```
<form oninput="x.value=parseInt(a.value)+parseInt(b.value)">
    <input type="number" id="a">+
    <input type="number" id="b">=
    <output name="x" for="a b"></output>
```

```
</form>
```

This syntax does three things:

1. It defines two variables that the user enters: a and b.
2. It defines that the operation to perform is addition.
3. It defines a box where the output goes.

This example looks like this in a form (with values entered):

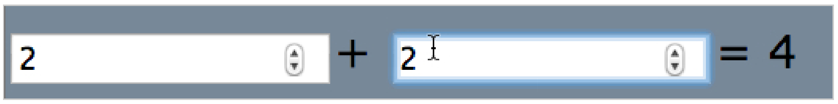


Figure 21 form output adding 2+2

Note:

You may have noticed that the output element is within a <form> element. The calculation takes place without a form being "submitted," and it takes place in a form of its own. Therefore, we'll place this element under the close </form> element for our main form. This will serve as a handy calculator on the page, but it won't be part of the same <form> element that users submit.

Let's customize the example to help a user quickly calculate how much money they might donate each month to our website project. Insert this code in your form, after the close </form> tag for the existing form:

```
<p><input type="submit" value="Click here to submit the form"/>
<input type="reset" value="Click here to reset the form"/></p>
<p><label for="oninput">How often do you visit this site monthly?</label></p>
<form oninput="x.value=parseInt(a.value)*parseInt(b.value)">
<p><input type="number" id="a" placeholder="Visits/day"> * <input type="number"
id="b" placeholder="Days you visit"> = <output name="x" for="a b"></output>
</form></p>
```

Output should be like below:

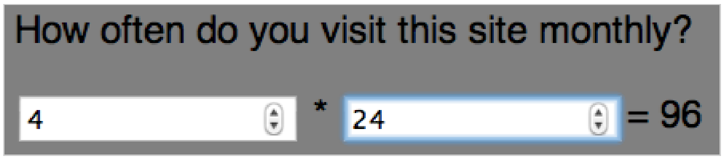

How often do you visit this site monthly?

Figure 22 Customizing an output element to multiply two user-entered values

In customizing the form, we tweaked the example. We changed the mathematical operator from "+" (add) to "*" (multiply). And we added a label and placeholder text. The placeholder text makes this form much easier to understand. It looks like this before a user starts entering values:

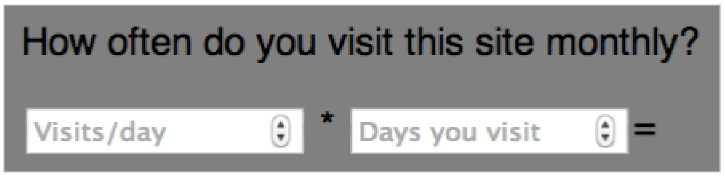


Figure 23 Using placeholder text with an output element

Our form is getting friendly! We provided a helpful hint in the Name field and a list of pre-typed options for the Country field. But there are even more powerful form properties in HTML5. Let’s explore them!!

Understanding HTML5 Field Types

HTML5 field types make it easy (and fun) to enter values in a form, and they validate form input. Before we get to the fun part, let's quickly scan a list of HTML5 field types:

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

These field types correspond to the type of data you're collecting. The color field type, for example, displays a color picker for users. The date field type lets visitors choose a date from a pop-up calendar. Not all HTML field types are that interactive, but all of them, in one way or another, make it easier for users to input data.

The basic syntax is:

<input type="type" name="name">

Let's look at two of the best examples of how HTML5 displays intuitive, helpful (and yes, sometimes fun) ways to enter different kinds of data: date and color. Those of us who are past a certain age dislike having to reveal our birthdays. But that can be painless with HTML5 field types. Here's an example of collecting that information with an HTML5 date field type:

```
<p><label for="date">What's your birthday?</label> <input type="date"
name="birthday"></p>
```

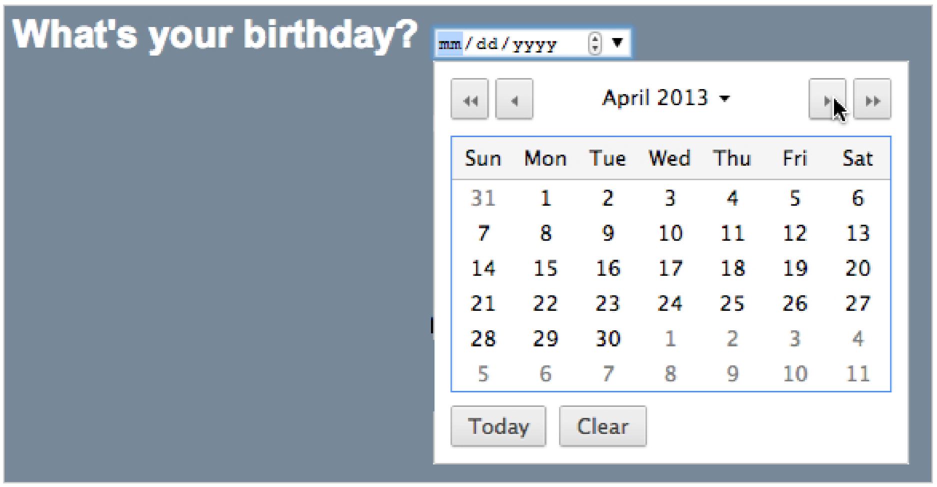


Figure 24 Collecting a date in an HTML5 form

How do you ask someone to provide a color? HTML5 makes it easy to collect accurate color data, and I think this qualifies as fun as well. Visitors can use a variety of color systems to enter colors if they know a color value in RGB values. Here's an example of a color field type combined with a date field type:

```
<p><label for="color">Please choose the color shirt you wish to order</label>
<input type="color" name="shirt-color"></p>
```

Output:

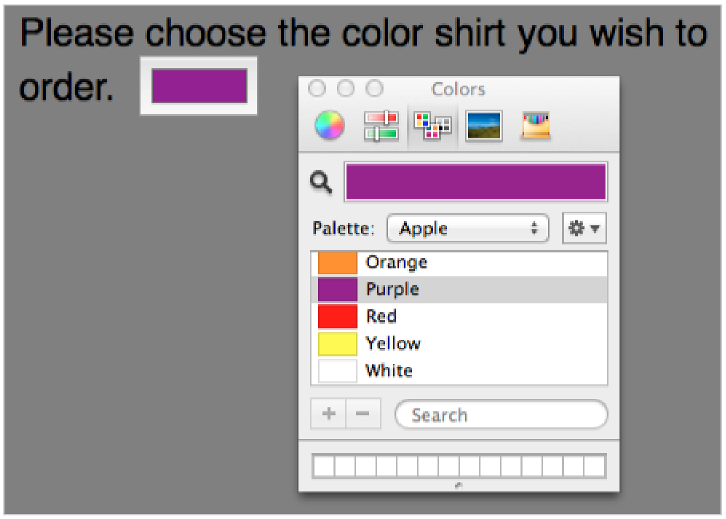


Figure 25 Collecting a color value in an HTML5 form

Everybody makes mistakes, and some users will probably make errors even when answering simple questions like these. Let's talk about what you can do to painlessly correct these errors.

Why Validation Matters

HTML5 form fields have built-in validation properties. In other words, they check data before submitting it and alert your visitor if there are problems. Before we examine how those work, let's step back for a second and survey what validation is and how it works.

Form validation means testing form data before that data goes to a server. For example, what if we're asking a user to tell us the year in which they born, and the user enters a value of 25? The user probably misunderstood and entered their age instead of birth year. A validation test might require that the field have a minimum value of 1900 to screen out entries that don't seem to make sense. There are three ways to validate form data:

- With JavaScript written to test form field values
- With server-side scripts that test form field content when the form goes to the server
- With HTML5 validation properties

All these techniques have advantages and disadvantages. JavaScript validation works in old versions of Internet Explorer. Server-side scripts also work in any browser, but they're slower since the data has to go to a server for validation.

HTML5 validation is much easier to implement since it doesn't require any scripting beyond our HTML5 coding skills. Also, it's fast, since validation takes place in a browser.

Adding Required Fields

The simplest and perhaps most useful HTML form field validation property is ***required***. Here's an example of applying the required parameter to the Name field in the form we've been working on:

```
<p>
   <label for="name">Name</label>
   <input type="text" name="name" size="32" required>
</p>
```

Adding a required property forces user to enter something in a field. If they don't, they see an error message like this after submitting the form:

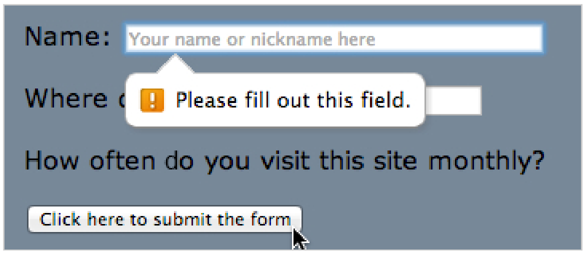

Name:
Your name or nickname here
Where c
Please fill out this field.
How often do you visit this site monthly?
Click here to submit the form

Figure 26 An error message indicating that a field needs filling out

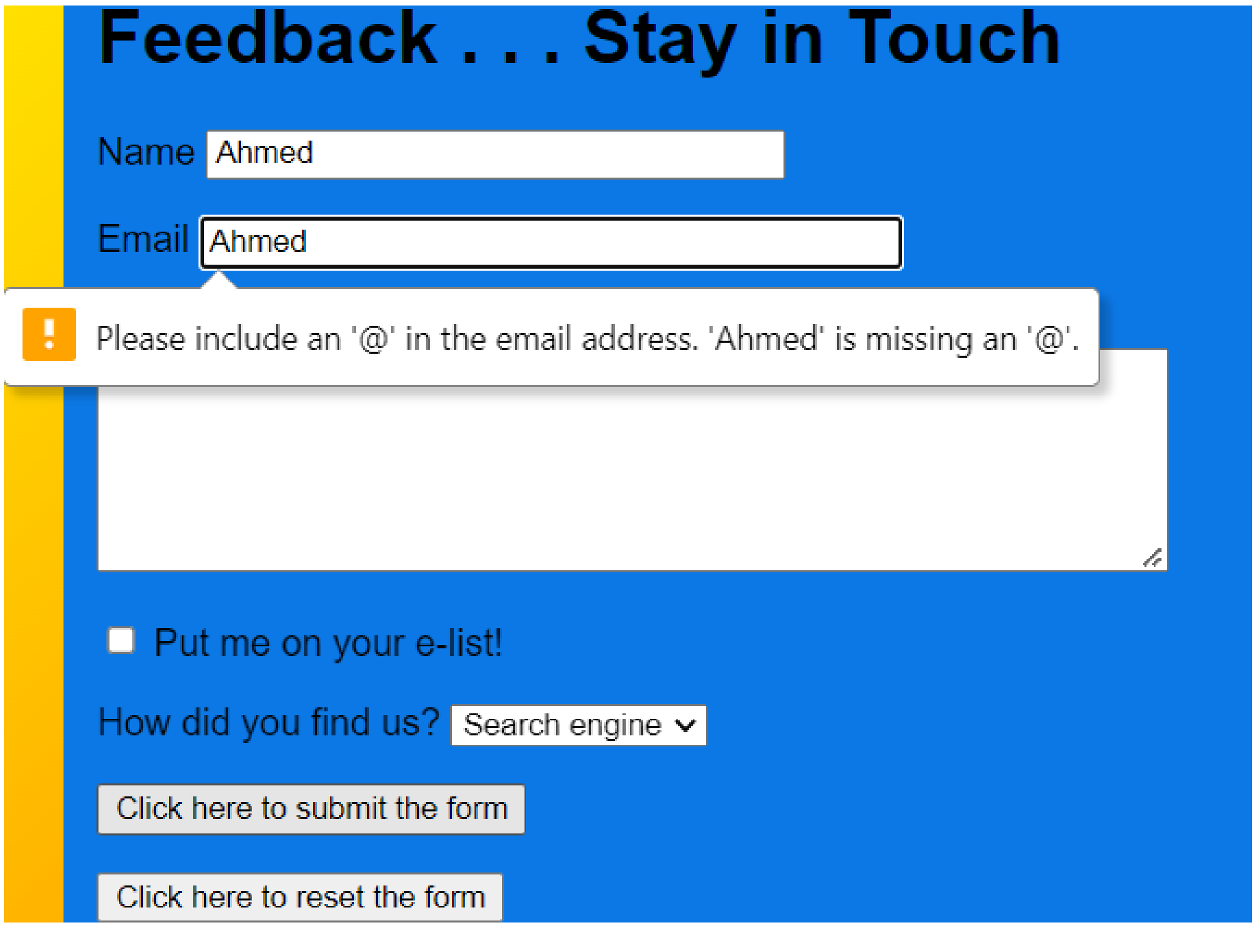
I'm sure you can see how helpful the required property is. Using it means you're much less likely to receive forms that are missing crucial information, such as credit card numbers or shipping addresses.

Using Email Validation

We've already talked about how the HTML5 color and date field types provide user-friendly tools for entering data. Other HTML5 field types don't have any special display properties; they're mainly for validation. The email type, for example, requires that anything the user enters in that field must look like an email address. You can test this by adding the following after the name field:

```
<p>
  <label for="email">Email</label>
  <input name="email" type="email" size="40" required>
</p>
```

Then test the field by entering something that doesn't look like an email address and clicking Submit. Error messages vary by browser, but your result should look something like this:


Feedback . . . Stay in Touch

Name: Ahmed

Email: Ahmed

Please include an '@' in the email address. 'Ahmed' is missing an '@'.

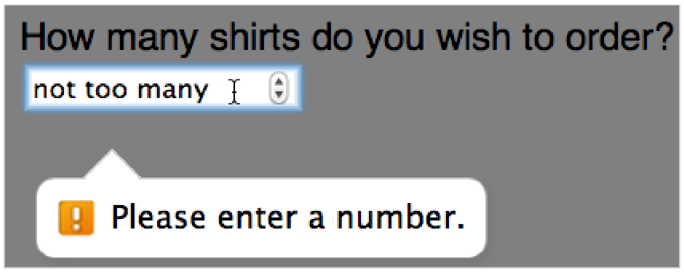
Put me on your e-list!
How did you find us? Search engine
Click here to submit the form
Click here to reset the form

Figure 27 An error message indicating that the content doesn't look like an email address

Including Number Validation

Let's take another example—number validation. The number type field will accept number values only (written with numerals, not text like "twenty-five"). To define a number type field for validation purposes, add this code after the email field:

```
<p><label for="quantity">How many shirts do you wish to order?</label>
        <input type="number" name="quantity"></p>
```


How many shirts do you wish to order?

not too many

Please enter a number.

Figure 28 An error message indicating the content of a field isn't a number

Save this form as form2.html.

Managing Form Data with Server-Side Scripts

So far, we used the simplest way to collect form data. The mail-to action sends form data to an email address by launching the user's own email client. More complex and more professional techniques for managing data rely on server-side scripts. By server-side, I mean the scripts are on your hosting server. Data goes to the server for processing using programs that run in the server, not in the user's computer.

Server-side scripts range from simple to complex. I mentioned a few server-side scripting languages in the lesson earlier—they include Ruby, PERL, and ASP. Server-side scripting is beyond the scope of this lecture. Instead, we'll use a helpful, free online resource to generate a server-side script.

Generating HTML and PHP at TheSiteWizard

There are a number of online resources for generating server-side scripts and matching forms. These resources always provide HTML for a form and PHP for the script because the form elements have to match the scripting, or the data won't process properly. In other words, if your form has a field named "email," the script needs to know that. The forms and scripts available from TheSiteWizard are editable, but we'll stick with the default settings. That means we'll get an HTML form from TheSiteWizard that collects feedback and serves as a signup form. That form includes tables—a design technique that advanced Web designers avoid, but it's something we can live with. And if you wish, you can edit the HTML from TheSiteWizard to replace tables with div tags for design.

Creating the Three Files We Need

Like many useful PHP scripts, the one we get from TheSiteWizard will require three HTML files:

- A feedback.html file that will hold the form
- A thankyou.html file that will display if the form data submits properly
- An error.html file that will open if there are validation issues with the submitted form (the PHP script will run its own validation on the server)

So, let's create those now. Use the following HTML to create and save a feedback file:

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="UTF-8">
```

```html
<title>Feedback</title>
<link href="style.css" rel="stylesheet" type="text/css">
</head>
<body>
<div id="wrapper">
<h1>Please share your feedback</h1>
</div>
</body>
</html>
```

Save the file as feedback.html

Save the below file as error.html

```html
<!DOCTYPE HTML>
<html>
<head>
<meta charset="UTF-8">
<title>Feedback</title>
<link href="style.css" rel="stylesheet" type="text/css">
</head>
<body>
<div id="wrapper">
<h1>Sorry, there was an error in your submitted form</h1>
<h2>Please press your browser's Back button to return to the form</h2>
</div>
</body>
</html>
```

Save the below file as thankyou.html

```html
<!DOCTYPE HTML>
<html>
<head>
<meta charset="UTF-8">
<title>Feedback</title>
<link href="style.css" rel="stylesheet" type="text/css">
</head>
<body>
<div id="wrapper">
<h1>Thanks for submitting your form</h1>
<h2>Click <a href="index.html"> here</a> to return to our home page</h2>
</div>
</body>
</html>
```

With those three files saved to your class site folder, we're ready to generate some PHP!

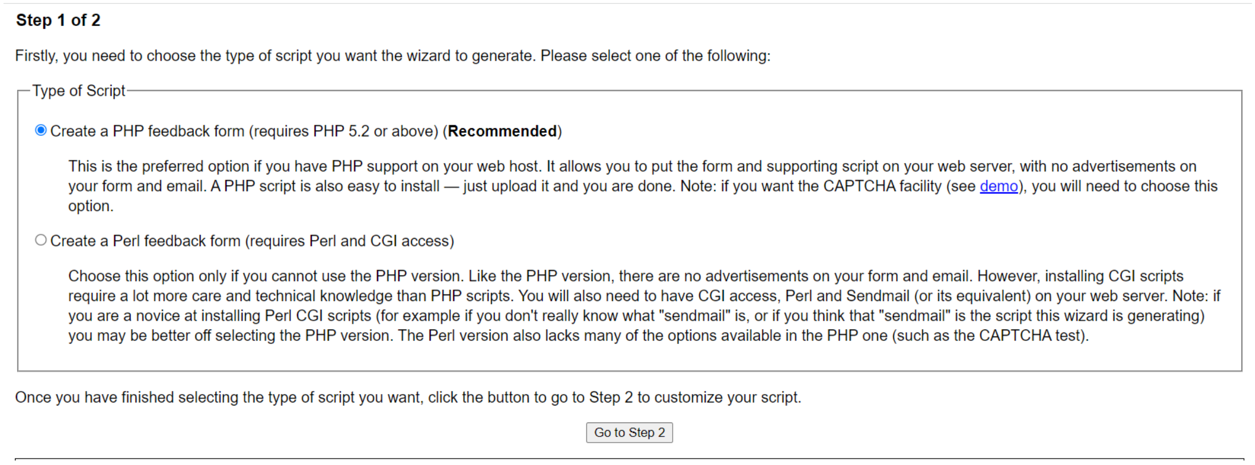
Note:

Before we generate some PHP, it's important to note the URL for your website. Your PHP file won't work on local computers. You have to upload it to your server in order to test it.

Generating PHP

Follow these steps to generate a form and a PHP file to manage form data:

1. In your browser, go to The Site Wizard Feedback Form (http://www.thesitewizard.com/wizards/feedbackform.shtml).
2. Scroll down the page to Step 1 of 2, and choose the radio button that says "Create a PHP Feedbackform (Requires PHP 5.2 or above)." The free hosting service that I recommended supports this version of PHP.



Step 1 of 2

Firstly, you need to choose the type of script you want the wizard to generate. Please select one of the following:

Type of Script

[✓ Create a PHP feedback form (requires PHP 5.2 or above) (**Recommended**)]

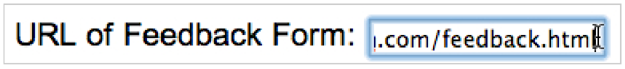
This is the preferred option if you have PHP support on your web host. It allows you to put the form and supporting script on your web server, with no advertisements on your form and email. A PHP script is also easy to install — just upload it and you are done. Note: if you want the CAPTCHA facility (see demo), you will need to choose this option.

Create a Perl feedback form (requires Perl and CGI access)

Choose this option only if you cannot use the PHP version. Like the PHP version, there are no advertisements on your form and email. However, installing CGI scripts require a lot more care and technical knowledge than PHP scripts. You will also need to have CGI access, Perl and Sendmail (or its equivalent) on your web server. Note: if you are a novice at installing Perl CGI scripts (for example if you don't really know what "sendmail" is, or if you think that "sendmail" is the script this wizard is generating) you may be better off selecting the PHP version. The Perl version also lacks many of the options available in the PHP one (such as the CAPTCHA test).

Once you have finished selecting the type of script you want, click the button to go to Step 2 to customize your script.

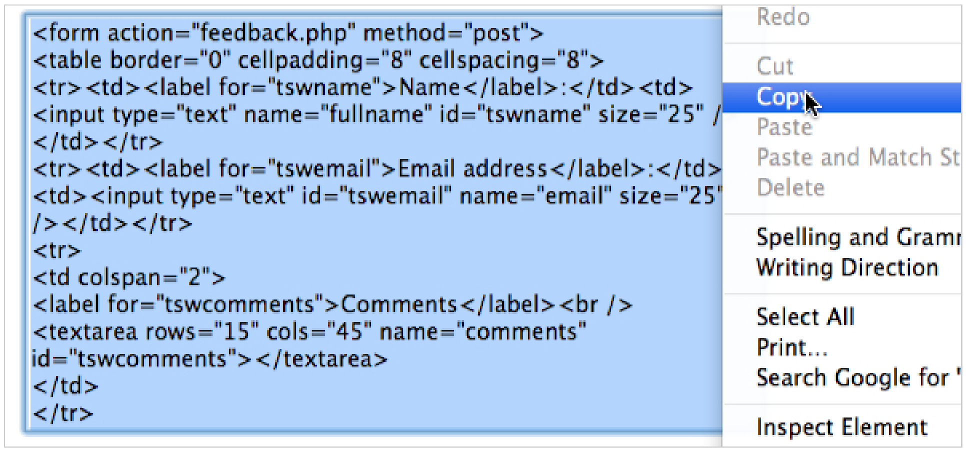
3. Scroll down a bit more, and click Go to Step 2. In Step 2 of 2, enter your email address in the Email Address field.
4. In the URL of Feedback Form field, enter the URL for your feedback.html page. Don't forget to use a full URL, starting with http://. Your URL will differ from mine, of course.



URL of Feedback Form: ı.com/feedback.html

Figure 29 Supplying the URL for the feedback page

5. In the Thank You and Error page fields, enter the full URLs for the thankyou.html and error.html files.
6. Feel free to survey the Advanced options, including instructions for using a Captcha code. But we'll skip those options here since the default settings will work well.
7. Scroll down the page, read the Conditions for Use, and then click Generate Script. The FeedbackForm Wizard: Results page opens.
8. Copy the Feedback Form Script code into your clipboard.
9. Create a new file in your code editor.
10. Copy the PHP code into that file, and save the file as feedback.php in the same folder you use for yourclass files. Feel free to explore the PHP file—it includes helpful and accessible documentation.
11. Back in your browser, at the Feedback From Script page, scroll down the page to "2. HTML Code."
12. Select and copy the HTML code.



```
<form action="feedback.php" method="post">
<table border="0" cellpadding="8" cellspacing="8">
<tr><td><label for="tswname">Name</label>:</td><td>
<input type="text" name="fullname" id="tswname" size="25" /
</td></tr>
<tr><td><label for="tswemail">Email address</label>:</td>
<td><input type="text" id="tswemail" name="email" size="25"
/></td></tr>
<tr>
<td colspan="2">
<label for="tswcomments">Comments</label><br />
<textarea rows="15" cols="45" name="comments"
id="tswcomments"></textarea>
</td>
</tr>
```

Figure 30 Copying the HTML for feedback form

13. Paste the copied HTML into your feedback.html fi le underneath this line of code: <h1>Please share your feedback</h1>
14. Save all four of the files we created: feedback.php, feedback.html, error.html, and thankyou.html.
15. Upload all four files.

Testing the Feedback Form and Script

To test the script, open your feedback.html page in a browser. Remember, you can't test this page on your local computer; you have to use the version you uploaded to the server. Try entering some feedback in the form. If its successful you will see Thank you message and if error you should see Error Screen.