How to Make a Web Site Part I: Understanding and Writing HTML

Adam Dachis — Everyone lives at least a little bit of their life on the web, and whether you develop web pages for a living, want to create a nameplate web site, or simply want more control over how your comments show up on web sites, having an understanding of HTML at your command is invaluable. With that in mind, in our first lesson on how to make a web site, we're covering the top-level basics of HTML—the predominant markup language of the web.

Nowadays it's easy to put together a web presence using social media and a personal landing page, but if you want to *actually* make your own web site you're going to need to learn HTML and CSS. Fortunately, we can help.

The video above will get you set up with a text editor, walk you through the basic structure of an HTML document, and introduce you to a few things about HTML you're going to want to know right away. By the end of the lesson you'll know how to create a basic HTML page. If you forget something or want a little additional reference material, check out the text below. It'll provide you with the basic information about HTML that you'll need.

What You'll Need

A plain text editor to write your HTML

Your web browser of choice to test your HTML

A desire to beef up your knowledge of how the web works

As you can see, you really don't need much to get started.

Getting a Plain Text Editor

In order to write HTML, you need a plain text editor. This video uses a plain text editor calledTextmate, but that'll cost you. If you want something free, you've got a lot of great options, including Notepad++ (Windows), Kod (Mac), or Sublime Text for either Windows and Mac. There are several other text editors out there, so you can use whatever you want so long as it is a *plain* text editor. While the term plain text is kind of a misnomer (here's why), it's basically used to describe text that doesn't have any style and is simply letters, numbers, and symbols. This means no bold, no italics, no different sizes, etc. While your plain text editor may have syntax highlighting—a feature that changes colors of the text based on what you type to make your code easier to read—this is just something you'll see in your text editor and something that is not saved into the file. Basically, you need a text editor that doesn't save anything but the text (which means you don't want to use something like Microsoft Word).

Note: For the purposes of this lesson, you should save all the files you create in your text editor with `.html` *as your file extension—for example, "my_first_web_page.html". You can edit a .html document in your plain-text editor of choice, but you can also view it in your browser. What you see when you open it in your browser will be very different than what you see when you open it in your text editor.*

What Is an HTML Document?

If you read the above section, you may have guessed that HTML—

which stands for HyperText Markup Language—is just a bunch of text saved as a document type that your browser identifies as using HTML. If you see an image on a web page, it's simply referenced in the text of an HTML document and not physically included as part of the file. All an HTML document really does is provide a set of text-based instructions that a web browser can interpret. It does this by encapsulating the page's text in tags, which we'll learn more about in a minute. It also uses these tags to tell the web browser to do things like display images, add line breaks, and more. Going further, HTML can be styled using CSS—which stands for cascading stylesheets—which we'll learn about in the next lesson. For now, just understand that HTML is a set of instructions for your browser that *you* are going to write.

What Are Tags?

Tags are used in HTML to specify certain elements on the page so the web browser knows how to render them. Here's what a set of tags look like:

```
<html> </html>
```

The above tags are the HTML tags. Your entire HTML document goes inside of those tags. All tags start with a less than symbol and end with a greater than symbol. They're called tags, in part, because those symbols make them look like tags. The starting tag simply has the term HTML inside of it, but you'll notice that the ending tag has a / before the term HTML. The / is what designates it is the closing tag. This tag tells your web browser that the first HTML tag is the start of the HTML document and the second /HTML closing tag is the end. Most tags look like this. For example, if you want to make text bold you might see this:

```
    <b>Bold text!</b>
```

Note: There are other ways to make text bold, too, so be sure to watch the video for a full explanation as these differences can sometimes be very important.
You'll also see tags that look like this:

```
    <img src="image.jpg" width="640" height="480" />
```

The above tag is an image tag. You've probably figured this out already, but its job is to display an image. There are two noteworthy things that are different about this tag. First, it doesn't have an ending tag. This is because the img tag is a self-closing tag. It doesn't need an ending tag because there is nothing that would go between a starting tag and an ending tag. You'll notice a / at the end of the img tag, however, and that's to designate the end. Back in the early days of HTML you didn't need to add that / to an image tag, and technically you still don't, but it's proper form. The other difference you'll notice is that the tag has a bunch of attributes. Attributes are things like src="" and height="", and they contain information describing more about the tag; in the case of the img tag, the source (src) attribute is always necessary.

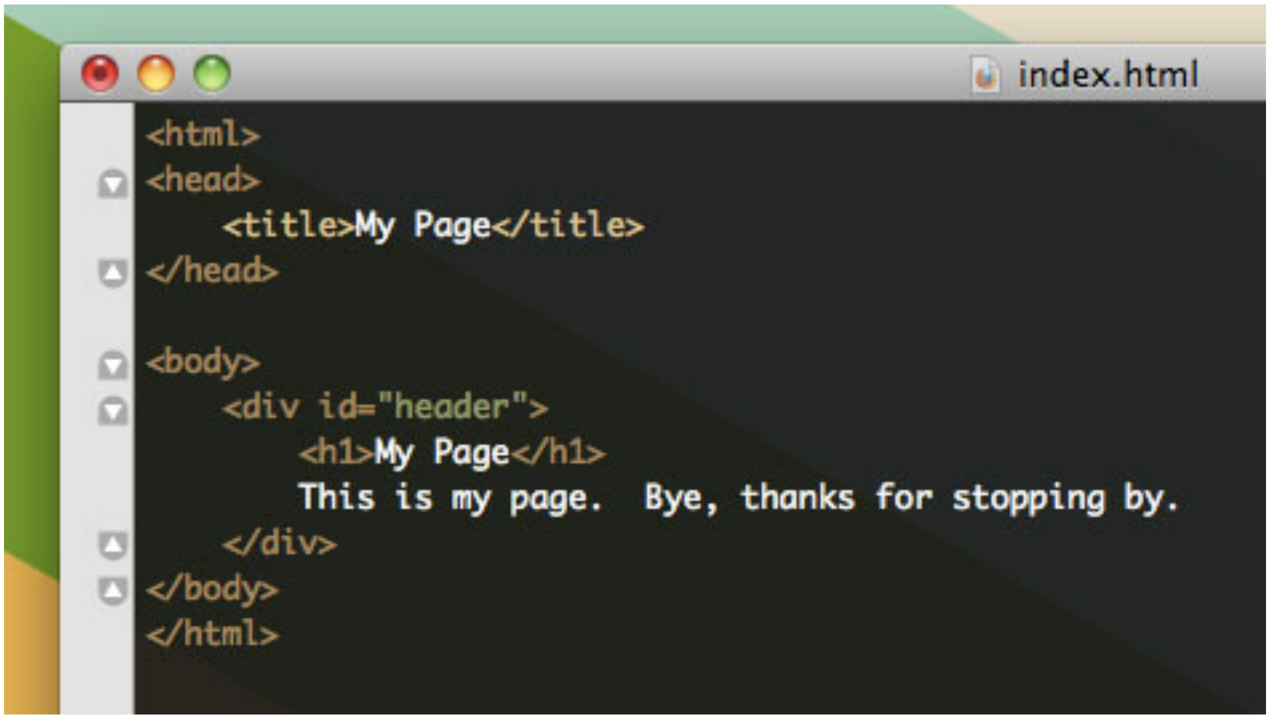
The src attribute specifies that the image file we want to display is image.jpg. Because we're just listing the file name, the browser will assume we're keeping that image file in the exact same location as our

HTML document. If you had a folder called images in the same place as your HTML document and kept the image in there, you'd set src to "images/image.jpg" because the / designates that we're going into a folder. If you wanted to load an image from an external web site, you could just but the full URL to the image (e.g. http://website.com/image.jpg). The other attributes simply specify the height and width of your image. Only the src attribute is required for the image tag, but if you don't specify the height and width of your image the browser won't know how much space to leave and it'll keep readjusting the page as it load. This looks kind of weird, so it's always better to specify the height and width in your img tags.

For a quick reference of some of the basic tag elements you can use in your document, this cheatsheet is a good place to start.

The Structure of a Basic HTML Document

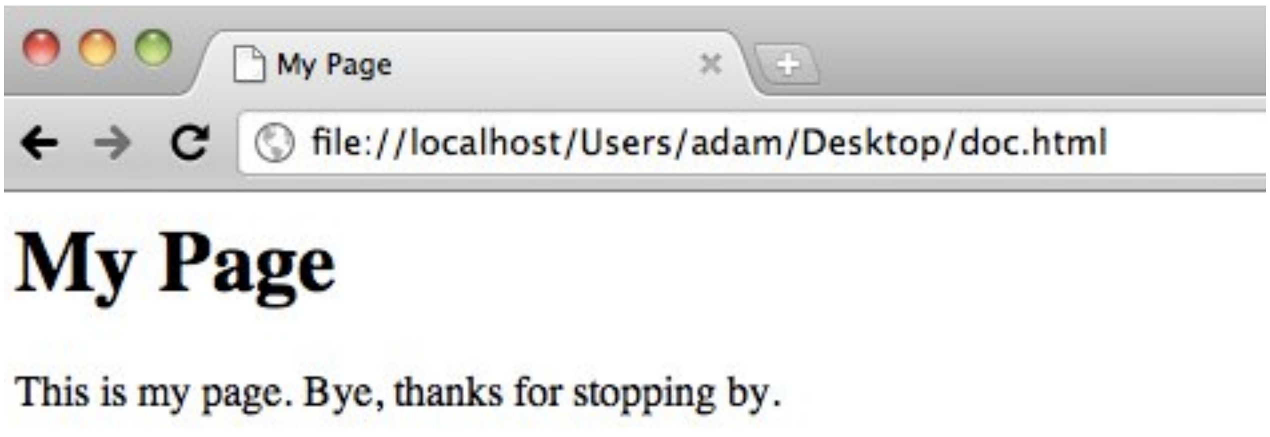
Now that you've got basic tag structure down, let's take a look at a basic HTML document's structure. You should know this is a very basic look and doesn't include absolutely everything you'll probably find in a fully developed HTML document, but it works just fine and keeps things nice and simple. Here's the very basic structure:



```
<html>
<head>
    <title>My Page</title>
</head>

<body>
    <div id="header">
        <h1>My Page</h1>
        This is my page.  Bye, thanks for stopping by.
    </div>
</body>
</html>
```

You'll notice that inside the HTML tags are HEAD and BODY tags. The HEAD tag encapsulates information that's not necessarily going to directly display on the page, such as the page title (which shows up as the window or tab title on your web browser), CSS styles, and other metadata. The BODY tag encapsulates information that *will* display on the page—your text, images, and rich media. The resulting HTML document opened in your web browser (just double-click the saved file or drag it into a browser window) will look like this:



My Page

This is my page. Bye, thanks for stopping by.

As you can see from the example, the body tag has a few things inside of it. First there's a DIV tag with an ID of header. DIV tags are used to contain bits of content so you can style them and move them around with CSS. You'll learn more about this when we dive into CSS in the next lesson.

Inside the header DIV tag is some text. The first part of the text is inside an H1 tag. H1, H2, H3, H4, H5, and H6 tags are all used to create header text. H1 is the largest and H6 is the smallest. By default they result in bold, larger text, but you can style them however you like using CSS. Because this HTML document has no CSS style information, the H1 tag will make the text My Page look bold. This effect is very similar to the big section title text you see in this post.

Below the H1 text is just some regular, un-styled text. Most browsers render un-styled text in the Times New Roman font at a size of 12pt. When you start adding CSS styles you'll be able to style this text however you like by setting a default style.

How to Make a Web Site Part II: Styling and CSS

Adam Pash — In part the first of our Night School series on how to make a web site, we learned a little about the bones of a web page: HTML. In today's lesson, we're going to start putting the clothes on our site using styling and CSS.

The video above walks you through the three ways you can apply style to elements in an HTML document. It assumes you've already watched the first lesson, so if you haven't, you may want to have done that first. By the end of this lesson, you'll understand how to add style properties to elements within your web page. For extra reference material, check out the text below.

What You'll Need

- A plain text editor to write your HTML
- Your web browser of choice to test your HTML
- A desire to beef up your knowledge of how the web works

As you can see, it doesn't take much to get started.

What Are Styles and CSS?

```
body {
   margin: 4px;
   border: 3px dotted #
   font-family: sans-serif;
   color: #000000;
   background-color: #FFFFFF;
}

h1 {
   padding: 5px;
   margin: 10px;
   border: 1px solid #C0C0C0;
   color:#FF0000;
   background-color:#0000FF;
}
```

CSS stands for **C**ascading **S**tyle **S**heets, and it's the language your browser uses to interpret how elements on a web page should look. The CSS language is pretty easy to understand: You define specific CSS properties, which span everything from element dimensions (width and height), text (weight, style, font-type, color, etc.), positioning, and spacing (margins and paddings).

Essentially, if it has anything to do with how an element should *look*, it has to do with CSS. To extend on the metaphor above, HTML and the elements in your document are the bones of the site. They provide structure. CSS dresses your site and makes it look nice—or not. That part depends on your design chops.

Written out, for example, CSS looks a little something like this: If I wanted to add a color to a piece of text, the corresponding CSS property is, unsurprisingly, `color`. To define the color property, you'd simply type `color`, a colon, declare what the style should be, then end with a semicolon. So, for example, `color:red;`

How to Apply CSS to Elements

Of course, you need to know where to apply your CSS styles so that the elements on your page reflect them. (You couldn't just go typing `color:red;` willy-nilly and expect results. You can use CSS to apply styles to any element on your web page in one of three ways:

> **Inline styles:** This method is best for quick, one-off styles that you want to apply to one element without a lot of overhead. You can add inline styles to any element by defining the style attribute of an element. For example:`<p style="color:red;">The text inside this paragraph tag will be red.</p>` You can define more than one property using inline styles as long as you remember the semicolon between them: `<p style="color:red; font-family:Helvetica; font-size:20px;">The text inside this p tag will be red, 20-pixels wide, and use Helvetica.<p>`
>
> **Internal stylesheets:** Inside the document (often inside the head), you can define styles for elements in the page using **selectors**. Internal stylesheets look like this:`<style type="text/css">`
>
> ```
>         h3 {
>                 color:red;
>                 font-family:Helvetica;
>                 font-size:20px;
>         }
> </style>
> ```
>
> In the example above, the `h3` is the selector. It's saying, "All H3

elements on this page should use the following styles." You can also define selectors using classes or element IDs.

External stylesheets: These stylesheets move the CSS to an external file ending with .css (for example, `style.css`. The syntax for external stylesheets is exactly like what you saw with internal stylesheets, but you don't need the style declaration. Instead, you link to your external stylesheet like so:`<link href="stylesheets/style.css" media="screen" rel="stylesheet" type="text/css" />`

In this instance, I've got the `style.css` file inside a stylesheets folder, and I'm telling the web browser: "When you load the page, looks for the CSS here." Everything inside looks the same as what goes inside the style tags of internal stylesheets.

When to Use Classes, IDs, and Element Selectors?

Most of the time, for the sake of organization, you'll want to use external stylesheets. But how do you decide which selectors to use? Let's quickly run through your options:

Element selectors: You should use an element selector when you want *most*elements of that type to look the same way on your page. So if you wanted all h2 elements to be 30 pixels, you could add the following to your CSS file:`h2 {`

```
        font-size:30px;
}
```

From then on, every h2 will respect that style unless you override it.

Class selectors: Classes are useful for broad styles that you want to apply to more than one element, but it may be a style that you want to use a little more selectively than you would when using the element selector. In the video, I defined the very boring `red` class like so:

```
.red {
        color:red;
}
```

Using the dot (.) before the text "red", I've told my browser that this selector is a class I'm creating. Now, in any element in my page, I can add an attribute of class="red" and its text will be red.
ID selectors: Defined in CSS by a preceding hash (#), the ID selector is for one specific element and one element only. Essentially this provides a way to adjust styles for one specific element without using inline styles—keeping all your styling in one place. (IDs are also very useful for JavaScript, but that's well beyond our scope here.) If I wanted an 800x600px div with a red background called my_red_div, it would look like this:

```
#my_red_div {
        width:800px;
        height:600px;
        background:red;
}
```

Watch the video to see all this in practice. Reading is one thing, but it all starts to make more sense when you can see it.

That's It?

In a nutshell, yes! Of course there's more to CSS, and there are some seriously more advanced ways of handling CSS selectors so you can really narrow down how elements inherit their styles, but that's really the basics. The only thing missing: the properties!

I'm not, however, going to walk through every CSS property available (we frankly don't have time in this lesson). As with most things of the programming nature, you'll learn more from searching out your options than I could provide in one quick guide. Here are a few CSS references you might want to browse:

- W3Schools CSS Refrence
- Added Bytes' CSS Cheat Sheet
- A List Apart

I'd love to hear more favorites in the comments for reference tools! I'm mostly a googler for this kind of stuff. I'm not CSS expert, but I've spent a good amount of time hacking CSS for my own pet projects, and believe me: A little bit goes a long way, but there's plenty you can learn if you want to go deep.

How to Make a Web Site Part III: A Site from Start to Finish

Adam Dachis — Now that you've got the basics of HTML and CSS down, we're going to take a look at how to actually use that knowledge to make a real web site. Today, we'll take you through the process of creating a site from start to finish.

★ This lesson presumes you know how to put together a site layout in Photoshop (or some other application). If you don't, fortunately we've covered that already so you can catch up.

Creating a web site isn't just about coding. You (or someone else) needs to have some basic design skills. You need to know how to prepare graphics for the web. You also need to know a few basic things about web servers to deploy your finished site. We've broken it all up into the four big tasks and take you through the entire process.

By the way, if you want to play around with the source code for our demo web site, you can download it here.

Design

MAKING A WEBSITE FROM START TO FINISH

01 DESIGN 02 PREPARATION 03 DEVELOPMENT 04 DEPLOYMENT

Design

We're not going to get into how to design a web site, technically or artistically. (<a href="http://lifehacker.com/#!5753625/basics-of-photoshop-designing-a-website">We've sort of done that already</a>.) You should have your site design figured out already, but there are a few things we do need to talk about before you start figuring out how to translate it into code.

First, the most important thing to know is that your font choices are sort of restricted online. While you can use the <a href="http://www.font-face.com/">@font-face</a> rule in CSS to externally load fonts, this isn't supported by older browsers. You also may need rights to use certain typefaces with this tag. That said, you <em>can</em> use @font-face to solve the problem of limited font choices on the web, but if you're not ready to jump into that world quite yet you should either use a web fonts service like <a href="http://www.webtype.com/">WebType</a> (which can be free depending on your use) or limit yourself to web-safe fonts. Which fonts are web-safe? Times New Roman and Arial are the most common options, but most operating systems come with several other built-in fonts that are considered web-safe. These include fonts like Tahoma, Verdana, Lucida Grande, Gill Sans, Trebuchet MS, Courier New, and Georgia. Do a search for web-safe fonts if you're looking for additional options.

Second, you need to consider what is going to be an image and what isn't. Nowadays you don't really need to use images for much more than complex graphics and photos as HTML and CSS can handle many of the complex things that we used to do with images. Menus, for example, can be created very easily in CSS with an unordered list. Generally you do not need text to be rendered as an image, but there may be some circumstances where you will need to do that (e.g. if the text is combined with a graphic).

Finally, you need to consider which images are going to be displayed as actual images or as backgrounds for one of your DIVs. How do you determine this? If you have text that's going to go on top of an image (e.g. with a menu), then you have your answer: your image will be a background. The reason this is important to know is because you need to export it unadorned with any text, images, or anything you're going to add later in the code. Once you've got that figured out, head on to the next step ("Preparation") where we discuss preparing your layout for coding and exporting any necessary images.

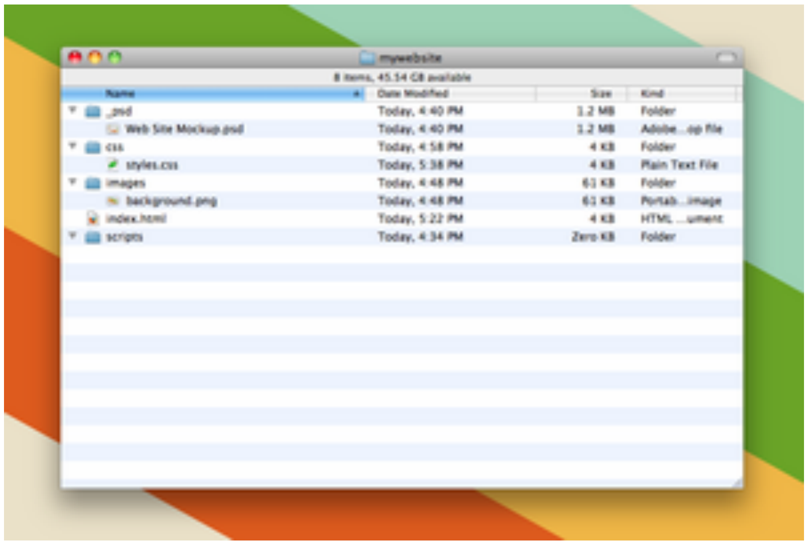
We're not going to get into how to design a web site, technically or artistically. (We've sort of done that already.) You should have your site design figured out already, but there are a few things we do need to talk about before you start figuring out how to translate it into code.

First, the most important thing to know is that your font choices are sort of restricted online. While you can use the @font-face rule in CSS to externally load fonts, this isn't supported by older browsers. You also may need rights to use certain typefaces with this tag. That said, you *can* use @font-face to solve the problem of limited font choices on the web, but if you're not ready to jump into that world quite yet you should either use a web fonts service like WebType (which can be free depending on your use) or limit yourself to web-safe fonts. Which fonts are web-safe? Times New Roman and Arial are the most common options, but most operating systems come with several other built-in fonts that are considered web-safe. These include fonts like Tahoma, Verdana, Lucida Grande, Gill Sans, Trebuchet MS, Courier New, and Georgia. Do a search for web-safe fonts if you're looking for additional options.

Second, you need to consider what is going to be an image and what isn't. Nowadays you don't really need to use images for much more than complex graphics and photos as HTML and CSS can handle many of the complex things that we used to do with images. Menus, for example, can be created very easily in CSS with an unordered list. Generally you do not need text to be rendered as an image, but there may be some circumstances where you will need to do that (e.g. if the text is combined with a graphic).

Finally, you need to consider which images are going to be displayed as actual images or as backgrounds for one of your DIVs. How do you determine this? If you have text that's going to go on top of an image (e.g. with a menu), then you have your answer: your image will be a background. The reason this is important to know is because you need to export it unadorned with any text, images, or anything you're going to add later in the code. Once you've got that figured out, head on to the next step ("Preparation") where we discuss preparing your layout for coding and exporting any necessary images.

Preparation


mywebsite

8 items, 45.54 GB available

Name	Date Modified	Size	Kind
_psd	Today, 4:40 PM	1.2 MB	Folder
Web Site Mockup.psd	Today, 4:40 PM	1.2 MB	Adobe...op File
css	Today, 4:58 PM	4 KB	Folder
styles.css	Today, 5:38 PM	4 KB	Plain Text File
images	Today, 4:48 PM	61 KB	Folder
background.png	Today, 4:48 PM	61 KB	Portab...image
index.html	Today, 5:22 PM	4 KB	HTML ...ument
scripts	Today, 4:34 PM	Zero KB	Folder

First, you want to measure your design. This is something you generally want to do *before* you begin your layout, but you need those specific measurements to start coding so you definitely need to know them now. You don't need to know every little measurement, but you need to know the measurements of your main layout structure. For example, if you have a left side of the page and a right side of the page, you should know the width (and height, if it doesn't vary) of each. You should also know the width of the spacing between them and the total width of everything combined. If your design is very much based on a grid and lines up with everything nicely, chances are your site's header will be this width as well. Basically, you need to know the dimensions of every major element on the site. You'll need to know the minor ones, too, but that's not really information you need to get started.

Once you've got your measurements we can start exporting images. In many cases you will not need to export very many images because you can recreate most of your design with HTML and CSS (and JavaScript, if you want to go there). For the few images you do need to export, however, you're going to need to choose between JPEG and

PNG. When your images are complex—like photos, or illustrations with many colors and a wide tonal range—you're generally going to be better off with JPEG from a file size standpoint. If you have simple images, or require that your images have a transparent background, you're going to need to save them as 24-bit PNG files.

★ If you're saving a 24-bit PNG from Photoshop, you'll need to hide the background layer before choosing "Save for Web" from the File menu so Photoshop knows to ignore it.

When you save your images you're going to need somewhere to put them. For this reason, and because you're going to have a bunch of other files once you start developing, you should start organizing your site now. If you were making a more complex site, I'd recommend structuring everything using the Model View Controller (MVC) architecture, but since we're just dealing with a simple site I'd recommend creating the following directories in your site's root directory:

css

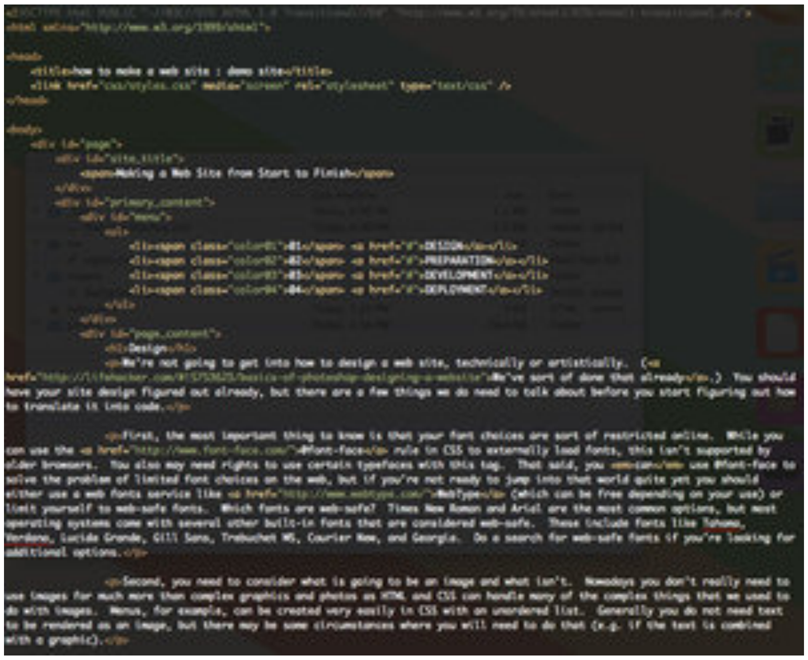
images

scripts

If it isn't obvious, you'll save your images in to the `images` directory and your CSS stylesheets in the `css` directory. The `scripts` directory is where you can store JavaScript files and other scripts you may end up using when developing your site.

In the root directory, you'll also want to make an `index.html` file in preparation for development. What is index.html? Let's say you only had one HTML file and it was called mydoc.html. If you uploaded that to your web server, which was hosting your site mywebsite.com, anytime someone visited mywebsite.com they would either get a directory listing of the files you've uploaded or possibly nothing at all. Using `index.html` as a file name tells the web server that it should display that file if someone goes to that specific directory. If you put an `index.html` file in your images directory, anytime somebody visited http://mywebsite.com/images they'd see a rendering of the HTML contained in that`index.html`. Basically, an `index.html` file is the default HTML to display so you're always going to need one.

Development



```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
    <title>how to make a web site : demo site</title>
    <link href="css/styles.css" media="screen" rel="stylesheet" type="text/css" />
</head>

<body>
    <div id="page">
        <div id="site_title">
            <span>Making a Web Site from Start to Finish</span>
        </div>
        <div id="primary_content">
            <div id="menu">
                <ul>
                    <li><span class="color01">01</span> <a href="#">DESIGN</a></li>
                    <li><span class="color02">02</span> <a href="#">PREPARATION</a></li>
                    <li><span class="color03">03</span> <a href="#">DEVELOPMENT</a></li>
                    <li><span class="color04">04</span> <a href="#">DEPLOYMENT</a></li>
                </ul>
            </div>
            <div id="page_content">
                <h1>Design</h1>
                <p>We're not going to get into how to design a web site, technically or artistically.  (<a
href="http://lifehacker.com/[illegible]/basics-of-photoshop-designing-a-website">We've sort of done that already</a>.)  You should
have your site design figured out already, but there are a few things we do need to talk about before you start figuring out how
to translate it into code.</p>

                <p>First, the most important thing to know is that your font choices are sort of restricted online.  While you
can use the <a href="http://www.font-face.com/">@font-face</a> rule in CSS to externally load fonts, this isn't supported by
older browsers.  You also may need rights to use certain typefaces with this tag.  That said, you <em>can</em> use @font-face to
solve the problem of limited font choices on the web, but if you're not ready to jump into that world quite yet you should
either use a web fonts service like <a href="http://www.webtype.com/">WebType</a> (which can be free depending on your use) or
limit yourself to web-safe fonts.  Which fonts are web-safe?  Times New Roman and Arial are the most common options, but most
operating systems come with several other built-in fonts that are considered web-safe.  These include fonts like Tahoma,
Verdana, Lucida Grande, Gill Sans, Trebuchet MS, Courier New, and Georgia.  Do a search for web-safe fonts if you're looking for
additional options.</p>

                <p>Second, you need to consider what is going to be an image and what isn't.  Nowadays you don't really need to
use images for much more than complex graphics and photos as HTML and CSS can handle many of the complex things that we used to
do with images.  Menus, for example, can be created very easily in CSS with an unordered list.  Generally you do not need text
to be rendered as an image, but there may be some circumstances where you will need to do that (e.g. if the text is combined
with a graphic).</p>
```

Now you develop your site using the skills you learned in the past two lessons and the additional HTML and CSS skills you'll acquire as you continue to learn web development. Since we've pretty much covered the basics of HTML and CSS, there isn't much else to say here. Rather than rehash the first two lessons, we're going to go over a couple of extra things you should know before you start developing your site for real and also provide you with the source code for the demo site we created for this lesson.

DocType

When creating your HTML document in lesson one, we focused on the basics of HTML structure but not some of the best practices. One thing you want to want to add to the very top of your HTML document is its DocType. You can learn a lot more about DocType on Wikipedia, but it basically is a document type declaration that says "these are the rules I'm following in my HTML document." This is important if you need to validate your code (to see if you made any mistakes), as the rules are a bit different for each DocType, but it also tells the browser what elements and structure to look for when rendering your HTML page. If you don't adhere to the DocType, it can cause problems, so this is something you'll want to explore further. For now, here's a quick look at what a DocType declaration looks like:

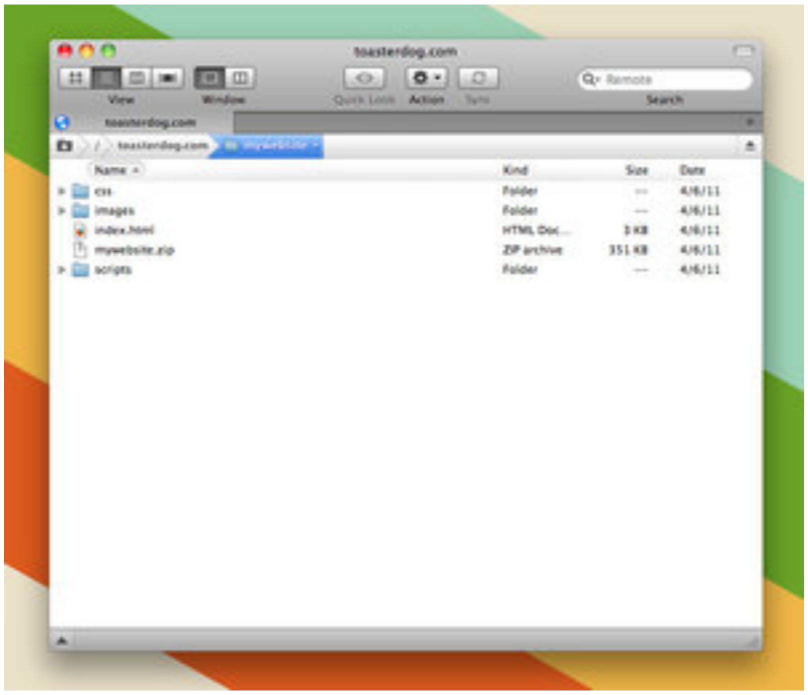
Check Out the Source Code

As you can see in the video, we used a lot of fun CSS techniques like drop shadows and rounder corners. If you want to explore these techniques further, download the demo site's source code and look around in the CSS. Once you have a grasp on the basics of HTML and CSS, the best way to learn new things is to look at the source code of other web sites. While you can't look at server-side code that was created in a server-side programming language like PHP, you can view

HTML source code simply by browsing to a web page and choosing "View Source" from your browser's "View" menu (unless it's in another menu, but it's generally in "View"). Big sites that have lots of content, such as Lifehacker, are not necessarily good models to look at. Much of the HTML code you see here is generated by server-side PHP code. Generally you'll learn more by looking at smaller sites, as the code is often static and/or the site is small enough that the amount of source code isn't horribly overwhelming.

Once you're done developing your site, it's time to deploy it.

Deployment



Name	Kind	Size	Date
css	Folder	---	4/6/11
images	Folder	---	4/6/11
index.html	HTML Doc...	3 KB	4/6/11
mywebsite.zip	ZIP archive	351 KB	4/6/11
scripts	Folder	---	4/6/11

Once your web site is finished, you need to deploy it. This often doesn't involve much more than uploading it to your web server. You'll generally accomplish this with FTP or SFTP, so you're going to need an application to do that. We're using Transmit in the video, but Cyberduck is a free file transfer app that runs on both Mac *and* Windows. All you need to do is use your file transfer application of choice to connect to your web server via FTP. From there, you just copy the files over to the web server and they should be immediately viewable on your web site.

Lifehacker Night School Part IV: Choosing a Host and Launching Your Site

Adam Pash — So you've learned the basics of HTML, brushed up on styling and CSS, and taken your web page design ideafrom concept to reality. Now it's time to find a host and launch your site.

Today's lesson is probably the easiest part of the whole make-a-web-site process (at least for the kind of site we're making), and it involves two simple steps:

- Find and register with a web host
- Upload your files to your host

Once you've done that, anyone with access to that wacky world wide web can see your site. Google can find it. Potential employees can see how talented you are. All the work you put into making it is now paying off!

But I'm getting ahead of myself. Let's talk a little bit about each step.

Step 1: Find and register with a web host

You can hem and haw all day about what web host to choose if you want. Choosing a host is like buying anything online: You do some research, find a few hosts that meet your needs, and maybe consider the recommendations of your peers before you commit. For a good starting point, take a look at our list of the five best personal web hosts. The reader-voted favorite of the bunch is DreamHost, and it's what I use for hosting various simple sites. I've tried a few alternatives over the years. Some were awful (Globat), some were just fine (Bluehost), but DreamHost has always been my favorite for simple hosting needs. That said, at something like $9/month DreamHost is more expensive than a lot of other cheap hosting plans. (I've also set up full web servers from scratch on virtual private servers at Slicehost and been very happy with them. Setting up a server on something like Slicehost is well beyond the scope of this guide.)

Note: Keep in mind that the kind of hosting we're talking about in this lesson isn't the same kind of hosting that could run a massive web site. Hosts like DreamHost are perfectly capable at hosting static pages (like we've made) or even low-traffic webapps that require a scripting language (like PHP) and a database.

Whatever service you choose, you'll need two basic things from your host:

- **A domain name:** This, as you probably know, is the human-friendly URL for your site. For example, when you visit http://lifehacker.com, that domain actually points to an IP address, which is a much less friendly string of numbers and dots. You want a friendly domain name.
- **A hosting plan:** This is the service that provides you with the disk space and bandwidth for your site. It hosts your files and transfers your web page to the computers that visit your site.

Notes on choosing a domain name

If you're putting together your nameplate site, consider trying to snag `yourname.com`. Whether it's based on your name or not, your luck securing the domain you want will vary depending on how unusual the name of the domain you'd like is. When I'm searching for domains, I've always liked using Instant Domain Search, a tool that returns availability of domains as you type. If you want to get clever with your name (del.icio.us-style), consider theDomain Hacks search tool. Finally, if you can't seem to secure anything, give Bust a Name a look. It suggests open domains based on synonyms of a few base words.

If you've never tried it before, here's an advanced warning: If you're not flexible, or you don't have a very unique domain name, finding an available domain name can be extremely frustrating. Be patient and play around, and eventually you'll find something you're happy with. (Remember, it doesn't have to be a .com domain—it's really up to your preference.) Once you've found an available domain you like, you're ready to proceed.

Buying your domain and hosting plan

You've got a choice at this point: You can either register your domain name with a domain registrar (like the popular Namcheap or another popular option) or you can simply register your domain when you buy your hosting plan. Many hosts—including DreamHost—give you a free domain registration when you register for hosting.

The catch with tying your domain name to your hosting plan is that often hosts aren't as good as registrars when it comes to easily managing, transferring, and handling your domains. To that point, I've been happy with DreamHost, and I'm sure most good hosts work fine as registrars, especially if you never have to transfer to another service. I was, by comparison, very unhappy with my first host, Globat, and how difficult it was to do anything with the domain I registered through them.

So make your domain and hosting choices. Once you have, you're ready to deploy your site.

Step 2: Deploy Your Web Site

This is the incredibly easy part. Since I have a DreamHost account, I used that in the video above, but whatever you decide on, the process should be nearly identical. Here's the broad version:

- Sign into your web host.
- Find the FTP (or for better security, SFTP) credentials. Basically you need an FTP, SFTP, or SSH username and password. On DreamHost, my first FTP user was created automatically; you can view, edit, or create new users here. If your host didn't create a user for you by default, you should be able to create one somewhere in your host's dashboard.
- Now open up your FTP client of choice. The free, open source Cyberduck (available for Windows and Mac) is a great choice if you're looking for something free and don't already have a favorite.
- Click the Open Connection button, choose the connection type (FTP or SFTP), enter your domain name, username, and password, and click Connect. If all your credentials were entered correctly, you should see a directory listing of files or folders, for

your domain, on your host.

. For the sake of this tutorial, I'm assuming you want your site in the root directory of your server so that when someone visits, for example, `yourname.com`, it will load the site you created. To do that, you need to find the root folder of your domain. On DreamHost, your initial directory listing will show all the domains you're hosting, so the root of, for example, my adampash.com domain is inside the adampash.com folder.

. Now that you've found your root, all that's left is to upload your site from your local computer to your server! There's not much to this part: Just drag the files from your desktop to the root folder you've opened in Cyberduck (or whatever FTP client you're using). Since your site is likely pretty lightweight, everything should upload in just a few seconds, and your site should be live for the visiting!

Congratulations! Believe it or not, that's all there is to it. You've successfully deployed your first web site.