Making Websites for Beginners

Clarissa Littler

July 21, 2016

1 Introduction

1.1 The point of this document

This note is a small tutorial covering the very basics of HTML and CSS, to accompany the corresponding two-hour lecture given through the Multnomah County Library system.

1.2 What we'll be covering

In this tutorial we'll be assuming that you have no experience with programming, markup languages, or creating websites.

We'll be covering

- **what** HTML is
- **how** you use HTML to create a site
- **what** CSS is
- **how** you use CSS to format the appearance of HTML

A companion tutorial found here includes an explanation of how to use JavaScript to dynamically modify and control both the HTML and CSS in the browser, which allows sites to be interactive. Similarly, a tutorial found here covers the basics of JavaScript.

This tutorial will be updated periodically and requests to

- add content
- fix typos
- clarify explanations

should be made through the issues page on the github repository.

1.3 What won't we be covering

Just as important as discussing what we will be covering is what we *won't* be covering. In this case we won't be providing

- an exhaustive explanation of every HTML tag
- an exhaustive explanation of every CSS property
- principles of proper graphic design
- accessibility best practices

These are all good, important, things to study when learning to make websites but they're beyond the current scope of this tutorial and its corresponding lecture.

1.4 What is a website?

Before we jump into describing how we make websites we need to describe what websites even are in the first place.

Every page you visit on the internet is made up of two sides: the server and the client.

The *client* is, generally, your web browser. Browsers are programs, such as Chrome or Firefox or Safari, that can process all the data that makes up a web page—the HTML, the CSS, the JavaScript code—and **render**, or display, it as an actual page you can see and interact with using your mouse or your keyboard or touchscreen.

The *server* is what sends all of this data to your browser, the client, and "serve" requests made by clients. These requests, which are formally called

HTTP requests, can be anything from “get me the website you run so I can see it” to “post this tweet” or “delete my account”.

In these notes we’ll be focusing exclusively on the **client** side of making a web page, which is writing the HTML and the CSS and the JavaScript, rather than the **server** side which is about handling HTTP requests and database management. Sometimes you’ll see the client side development called “front-end” and the server side development called “back-end”.

1.5 The pieces of a website

As we’ve discussed, there are three pieces of information that a browser needs to display a webpage to you. The first is the *HTML*, which is the main content of the site: the text, the buttons, the sections, the forms, the pictures. All the fundamental pieces of the page.

In contrast, the *CSS* is the look of the content: the colors, the fonts, the backgrounds, the sizes of things. The difference between HTML and CSS is like the difference between the script of a comic book and the art of a comic book. They’re both necessary and they both work together to tell your story, but they are fundamentally different pieces.

The third piece is the *JavaScript* code which allows your web pages to be interactive. The JavaScript code that accompanies a web page describes how the user interface of the page functions. For example, whenever you type into a search bar and start seeing results as you’re typing, that’s JavaScript code running in the background every time you type a key. Any time you’re on a web page and you see animation when you mouse over a menu, or elements of the page change color as you navigate through the site, or anything that is *dynamic*, that changes, when you’re interacting with the site you’re seeing the effects of the JavaScript code that’s running in the background.

2 HTML

HTML stands for HyperText Markup Language, now today we never speak about “hypertext”, but the term essentially describes web pages back when they were simple text documents with *links* (which was the **hyper** in hypertext) that directed you to different parts of the internet.

The *markup* in HTML means that you’re *formatting* text. A lot of people might be familiar with the idea of *formatting text* from the very simple markup some social media and commenting systems have, where you can put asterix (*) around your text and will be bold or slashes (/) right text to italicize it or other simple ways to change your text. These are examples of *markup* where you are modifying the text to signal something about it.

In this vein, HTML is a *markup language* for *hypertext*, or in more modern parlence a way to format web pages.

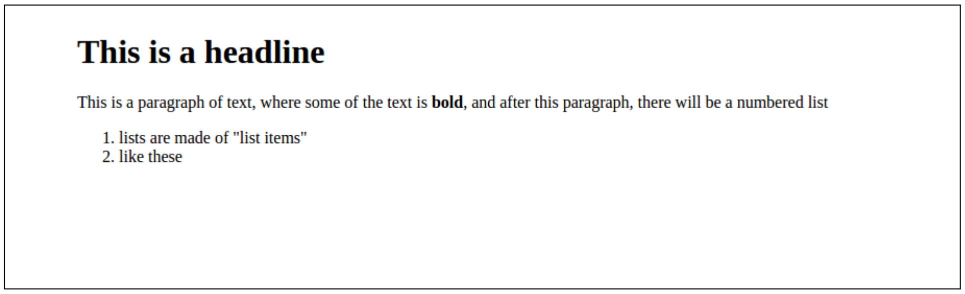
2.1 Tags

The most basic concept in HTML is the *tag*. Tags are how we markup text. Let's look at a simple example that will show that **all tags** have the same basic format:

```
<body>
  <h1>This is a headline</h1>
  <p>This is a paragraph of text, where some of the text is <b>bold</b>, and
    after this paragraph, there will be a numbered list
  </p>

  <ol>
    <li>lists are made of "list items"</li>
    <li>like these</li>
  </ol>
</body>
```

If you point your browser to the accompanying file `firstHTML.html` you'll see a very simple page that looks like this


This is a headline

This is a paragraph of text, where some of the text is **bold**, and after this paragraph, there will be a numbered list

1. lists are made of "list items"
2. like these

So every pair like `<body>` and `</body>` are **opening** and **closing** tags that markup the text and other elements within them. The *body* of our web page, which is the part we see, should be wrapped in `<body>` tags.

The other tags we see here are the `<h1>` tag, which is a *headline* tag. The *headline* tag denotes that the text "This is a headline" means something large and important, like a title, a section header, etc. This particular headline tag was `h1` but there's also `h2` through `h6`, which are progressively *smaller* headlines as the number *increases*. The smaller headlines give us subsection names and other ways of organizing your text.

The *paragraph* `p` marks all the text between the `<p>` and the `</p>` as a single paragraph.

The `ol` tag marks an *ordered list*, that is a list that is numbered like a list of steps or like an outline. It just creates an *empty* list, however, and you need to use the `li` tag to create items in each list. All the text between the `<li>` and `</li>` tags goes into a single bullet point.

Now, you might be wondering why we need these pairs of **opening** and **closing** tags, and the answer is that we need *The Right Thing* to be unambiguous to the browser. If we pretend we don't have closing tags it's not obvious what the following HTML should do:

```
<body>
  <ol>
    <li>This is a list
    <li>but
    <li> there's ambiguity here
  <ol>
    <li> where does this part go?
    <li> is it a sublist or a second list?
```

We see a second ordered list is started, but is this a nested list like you'd see in an outline or is it a second list altogether? Keeping in ending tags we can make ourselves clear, both to each other and the computer:

```
<body>
  <ol>
    <li>This is a list</li>
    <li>but</li>
    <li> there's ambiguity here</li>
  </ol>
  <ol>
    <li> where does this part go?</li>
    <li> is it a sublist or a second list?</li>
  </ol>
</body>
```

The last point that should be made in this section is that there's a distinction between *tags* and *elements*: tags are how you markup the text when you write HTML, but *elements* are what the browser creates after reading the markup: e.g. a pair of `<p>` and `</p>` tags becomes a paragraph *element* when the page is displayed.

This is a small distinction, and often people use element and tag interchangeably, but there are some distinctions in usage to be aware of when reading more about HTML and web-programming. For example, when using JavaScript to make a page interactive you'll almost exclusively see people describe interacting with *elements*, not *tags*.

2.2 Scaffolding

Before we delve more into the details of HTML, there are a few other preliminary structural things to describe.

First, it's best to start each HTML document with the special line `<!doctype html>`.

Second, all of your HTML needs to be wrapped up in `<html>` tags.

Finally, there is the *head* and the *body* of the HTML. The *head*, denoted by the `<head>` and `</head>` tags, is the place where you put information that won't be displayed in the page but you still need for the page like JavaScript code and CSS definitions. The body tags should be wrapped all of the content you're intending to display.

Essentially every HTML page you write should follow the follow template. It's also customary to end HTML files in `.html`.

```
<!doctype html>

<html>
  <head>
```

```
    ...
  </head>
  <body>
    ...
  </body>
</html>
```

2.3 Semantic markup

A phrase you'll frequently see when people discuss HTML is "semantic markup". Semantic, in this context, just refers to the idea of *meaning*. The idea is that all markup should be about signifying *something* about the text, about its meaning. That's why there are some tags that may seem redundant by the defaults of how they're displayed. For example, there's both a `<strong>` and `<b>` tags for text.

```
<!doctype html>

<html>
  <body>
    Here we have <b>some bold text</b>
    and some <strong>strong text</strong>.
    They look identical by default.
  </body>
</html>
```

By the default rendering of most browsers we can't tell the difference between them, but they *signify* different things: `<strong>` is used to emphasize text forcefully, the kind of thing where you'd speak loud and forcefully if you were reading it aloud, while `<b>` is meant to draw attention to a piece of text.

Other elements for semantic markup are elements such as `<article>` or `<section>`, that don't visually change anything but group other elements of the document together that are related to each other conceptually.

2.4 The basic elements

Now we'll go back to some of the basic elements that we need to cover in order to make a webpage.

First, we've already seen the basic *headline* element `<h1>`, which has variants of `<h1>` through `<h6>` for smaller and smaller sections. You will probably mostly find yourself using `<h1>` and `<h2>` to denote titles and sections the most, but if you're doing something more complicated like formatting a story or screenplay you might need even smaller headlines.

The file `headlineExample.html`, shown below, demonstrates the default relative sizes of the different types of headlines.

```
<!doctype html>
<html>
  <body>
    <h1>Big headline</h1>
    <h2>Smaller</h2>
    <h3>Smaller</h3>
    <h4>Even smaller</h4>
    <h5>Smallller</h5>
    <h6>Smallest</h6>
  </body>
</html>
```

As we've seen paragraphs should be formatted with the `<p>` tag.

Quotes are handled in a couple of ways, depending on whether we want a small inline quote or a big block quote, as in the following example, which is named `quoteExample.html` in the source directory of this tutorial

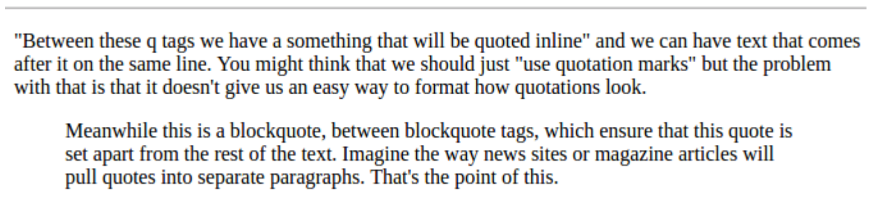
```
<!doctype html>

<html>
  <body>
    <p>
      <q>Between these q tags we have a something that will be quoted inline</q>
      and we can have text that comes after it on the same line.
      You might think that we should just <q>use quotation marks</q>
      but the problem with that is that it doesn't
      give us an easy way to format how quotations look.
    </p>

    <blockquote>
```

```
      Meanwhile this is a blockquote, between blockquote tags,
      which ensure that this quote is set apart from the rest of the text.
      Imagine the way news sites or magazine articles will pull
      quotes into separate paragraphs. That's the point of this.
    </blockquote>
  </body>
</html>
```

which, if you reduce the width of your browser window, will look something like



"Between these q tags we have a something that will be quoted inline" and we can have text that comes after it on the same line. You might think that we should just "use quotation marks" but the problem with that is that it doesn't give us an easy way to format how quotations look.

Meanwhile this is a blockquote, between blockquote tags, which ensure that this quote is set apart from the rest of the text. Imagine the way news sites or magazine articles will pull quotes into separate paragraphs. That's the point of this.

There's also basic ways to modify text: there's the `<b>`, `<i>`, and `<s>` tags which, by default, will make text appear **bold**, *italicized*, or ~~stricken through~~. Though, as above, we need to emphasize that tags should be used more for conveying meaning than for their default behavior.

- `<i>` is used to mean a different voice, while `<em>` is used for emphasis
- `<b>` is used for text that the eye should be drawn to, while `<strong>` is for text that is forceful
- `<s>` is used for text that is obsolete, replaced, but still kept to show the past revision

We've also seen basic lists in HTML, with `<ol>` being an *ordered* list (which is by default numbered) and `<ul>` being an unordered list.

Both of them have the same *list item* tag of `<li>`, as we can see in the following example

```
<!doctype html>
<html>
  <body>
    <ol>
```

```
      <li>This is an ordered list</li>
      <li>And here we have a nested list
        <ul>
          <li>and this is an unordered list</li>
          <li>which is by default</li>
          <li>a bulleted list</li>
        </ul>
      </li>
    </ol>
  </body>
</html>
```

The final basic element to discuss, and it's a very important one, is the *anchor* tag. The anchor, represented by `<a>`, is how you make *links* in HTML [1]. The most common use you'll have for the anchor is to link to other web pages. In order to do *that*, we need to introduce a little bit more syntax: attributes of elements.

In order to make a link to, say, the library's homepage you would write

```
<a href="https://multcolib.org">This is a link</a>
```

and what you'll get is the text `This is a link` in the form of a link that points to the library site. In this case, the `href` is the attribute name and `http://multcolib.org` is the value of the attribute. Every attribute follows this same basic format of `<tag attr_name="value">` in the opening tag of the element.

Attributes used to be used even more than they are now and they used to, before CSS became the standard technology for formatting, be used to set the height and width and other formatting properties of elements. Now, most of that is intended to be done in CSS and HTML should strictly be content only.

2.5 Block vs. inline

There's actually two kinds of elements that we've seen so far: elements that can be included *within* text and elements that take up space. The former are called *inline* elements and the latter are called *block* elements.

[1]There's also a `<link>` tag that you use in the head of the HTML, but it's used for including other files like CSS files into your HTML

So, for example, `<i>`, `<b>`, `<q>`, `<a>` and any tags that modify text *inline* are *inline* elements. Similarly, elements like `<p>`, `<ol>`, `<li>`, `<blockquote>`, and `<h1>` are all *block* elements.

The distinction may seem small but the main point is that you generally can only put inline elements inside block elements, but not the other way around.

2.6 Div and span

There's one last tool for "semantic markup" that we need to introduce before we move onto CSS and that's the `<div>` and `<span>` tags. They both perform very similar jobs, namely grouping elements together for the purpose of code and formatting. They do *nothing* visually by themselves.

The main difference between the two is that `<div>` is a *block* element and `<span>` is an *inline* element.

3 CSS

3.1 Style not substance

CSS, which stands for Cascading Style Sheets, is the component of our web page that controls how our page is displayed.

You can include your CSS in an HTML file directly by putting it between opening and closing `<style>` tags in the `<head>` element of your HTML.

A CSS file looks like a series of entries of the form

```
selector {
    property: value;
    property: value;
    property: value;
}
```

Selectors are how we choose what elements the entry applies to and we set their properties. There's three main selectors we'll talk about: by ID, by element type, and by class. There are other ways to select elements, but they're more advanced and, I'd argue, are features that aren't strictly necessary but rather allow you to do some things more conveniently.

For different kinds of elements there are different properties and we'll talk about only a few of them in this tutorial.

3.2 Selecting elements by ID

A very *common* way to modify an element in CSS is to use its ID. We haven't seen IDs yet, but it's a special attribute you can give elements. For example, in the following code we give a paragraph the ID of `para` and then use the CSS selector `#para` to choose that paragraph and give it a `color` property of `blue`.

```html
<!doctype html>

<html>
  <head>
    <style>
      #para {
        color: blue;
      }
    </style>
  </head>
  <body>
    <p id="para">This is the text within our paragraph.</p>
  </body>
</html>
```

IDs should be unique to a given element, with no two elements in the page having the same ID. If you find yourself wanting to apply the same styling rules to a number of different elements then you want to use a *class* instead.

3.3 Selecting elements by class

In the following example, we're going to define a *class* in our CSS that changes the `font-weight` to `bold`, the `color` to `red`, and the `width` to `200px`. We'll then give both a paragraph and an ordered list this class and see that it effects the text inside both of them.

```
<!doctype html>
<html>
  <head>
    <style>
      .ourClass {
        color: red;
        width: 200px;
        font-weight: bold;
      }
    </style>
  </head>
  <body>
    <p class="ourClass">Here's the text in one paragraph.
    There's going to be a fair decent length of text here so we
    can see that the width restriction causes the text to wrap around.</p>

    <ol class="ourClass">
      <li>Here's a list here that's also going to have an item
      with at least a moderately long single element
      in order to show the effects of the width property</li>
    </ol>
  </body>
</html>
```

Note that even though we applied the class `ourClass` to the ordered list, it still applied to the text *within the list item*. In general, properties defined in defined in cascading style sheets propagate down to all the "sub elements" within an element. So if you give an element a class, all of those properties will continue to propagate downward to every element *inside* that element.

3.4 Selecting elements by type

The final basic way to select an element in CSS is to select by *element type*. This means we can select, say, all paragraphs, all links, all lists, all divs, all spans etc.

Let's look at an example that modifies all of our paragraph elements, changing the background color and the font size.

```
<!doctype html>
```

```html
<html>
  <head>
    <style>
      p {
        font-size: large;
        background-color: green;
        color: blue;
        width: 200px;
      }
    </style>
  </head>
  <body>
    <p>Our first paragraph is here.
    There’s some text and things of that ilk.</p>
    <p>This is our second paragraph,
    beholden to no one but itself. A wild rebel of a paragraph</p>
    <p>Our third paragraph lies here,
    relentless in its comformity. There’s not much to say about ol’ thirdy,
    they’re simply stoic and resolute in their paragraphness.</p>
  </body>
</html>
```

3.5 Being specific

It’s actually possible to *combine* the selectors above to efficiently override style settings. We can help our rebellious second paragraph in the previous example to be **more** rebellious with a little rearranging of our code.

```html
<!doctype html>

<html>
  <head>
    <style>
      p {
      font-size: large;
      background-color: green;
      color: blue;
      width: 200px;
      }
      p.rebel {
        width: 300px;
        background-color: white;
      }
      </style>
  </head>
    <body>
      <h1 class="rebel">This time we also have a rebellious headline,
      which should be unchanged</h1>

      <p>Our first paragraph is here. There's some text and things of that ilk.</p>
      <p class="rebel">This is our second paragraph,
      beholden to no one but itself.
      A wild rebel of a paragraph</p>
      <p>Our third paragraph lies here, relentless in its comformity.
      There's not much to say about ol' thirdy,
      they're simply stoic and resolute in their paragraphness.</p>
      </div>
    </body>
</html>
```

If you look at this web page, which is in the file `combineSelection1.html`, you'll see that the headline isn't changed by having the rebel class but the *paragraph* with the rebel class is effected. There are, of course, other ways

3.6 Setting style attributes directly

One last topic is that it's possible to, essentially, shortcut the process of defining entries in a CSS file and, instead, set the CSS properties directly with the "style" attribute.

This is not the best way to use CSS in general, but it can be convenient for fine tuning or experimenting. If you have only a single thing that needs to be done in one particular place it can be useful. Like the following example

```
<!doctype html>

<html>
  <body>
```

```
    <p style="width: 200px; color: blue;"> So here we have a paragraph of text,
    like so many before,
    and we're setting the width directly in the style because
    we're only going to ever have a single paragraph.
    Also, at some point, we want just a single of piece of text to get
    <span style="font-size: 30pt">really big</span> for no discernable reason.</p>
  </body>
</html>
```

You can see that we still set the style in the attribute like we did before: semi-colon separated property value pairs.

3.7 Going deeper

How to make pleasing and useful websites in CSS is deceptively massive topic. The best thing I can recommend for learning more is to think of things you'd like to try and use resources like the Mozilla Developer Network to look up elements and CSS properties and attributes to see if there's a way to make what you want.

There are even more ways to do selection in CSS. We haven't even covered the topic of *psuedo-class selectors* which allow you to select things like the first child or whichever element is in focus.

4 Further resources

(this section will continue to be updated as I find resources I think are helpful)