First Steps

Let's learn some HTML!

Quick facts:

- HTML stands for Hypertext Markup Language. Invented in the early 1990's, researchers were looking for a way plain-text documents to contain "hyperlinks," a new invention to replace footnotes. Hyperlinks allowed readers to simply click a word to have the relevant cross-referenced document loaded, rather than having old-style footnotes at the bottom of the page or chapter.

- HTML has undergone many revisions since. We will be using HTML 5. Modern HTML provides a way to completely "mark up" a document's content to denote the start and end of headings, paragraphs, bullet lists, images, and more.
- Important terminology:
    - Element
    - Attribute
    - Opening Tag
    - Closing Tag
    - Content
    - Document
    - Head
    - Body

> See w3schools.com for a comprehensive HTML reference.

Elements

Open up your text editor and type this into a new file:

```
<h1>Hello!</h1>
```

Save your file and open it in your browser.

Ta da! You're a website developer!

THE BASICS

The code you wrote consisted of one HTML *element*. An element consists of:

- An *opening tag*, in this case `<h1>`
- *Content*, in this case `Hello!`
- A *closing tag*, `</h1>`

The element is identified by its opening tag. We would say that we wrote an "h1 element".

ATTRIBUTES

Elements can also contain *attributes*. Some elements require the presence of certain attributes in order to function. For example, let's add a link to Google:

```
<a href="https://www.google.com">Search</a>
```

Save the file and refresh your browser. You should now have a link that you can click to go to Google's home page.

The `href` attribute above is how the browser knew what to do when the content *Search* was clicked.

Attributes have a *name* and a *value*, always separated by an `=` sign, and the value is always enclosed in quotation marks.

In the above example,

- The attribute name is `href`
- The attribute value is `https://www.google.com`

Code Comments

Before we get too far, we need to learn how to add *comments*. A comment is a note to the developer. It is text within the source code that we do not want the computer to interpret. Every programming language has a concept of code comments, but the syntax required to denote a comment and set it apart from the rest of the source code is different in each programming language.

SYNTAX

In HTML, we indicate that a line of text is a comment like this:

```
<!-- This sentence will not appear to the user. -->
```

The special opening tag `<!--` marks the start of the text to be ignored by the computer, and ends with `-->` .

Comments can last for several lines:

```
<!-- This sentence will not appear to the user.
Nor will this.
Or this.
-->
```

BONUS: DISABLING CODE

The other main purpose of code comments is to temporary disable some HTML code, perhaps as part of a debugging strategy:

```
<h1>Hello!</h1>

<!-- <a href="http://www.google.com">Search</a> -->

<h2>Goodbye!</h2>
```

Try that in your browser to confirm which code will be interpreted and which will not.

BONUS BONUS: KEYBOARD SHORTCUT

Using comments to temporarily disable a line of code is so common, you'll want to do it often.

In Atom, you can use the menu option `Edit | Toggle Comments`. But hopefully you'll be using this technique so much that the keyboard shortcut will become second nature, `Cmd /` (on Windows, `CTRL /`).

HTML Template

Most HTML pages have the same skeleton structure. Here's a snippet of code that you can use in every new HTML page:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>My Page</title>
  </head>
  <body>

  </body>
</html>
```

Starting from this, you can then change the `title` content and add your HTML elements inside of the `body` .

BONUS: ATOM SNIPPET

If Atom knows that you're writing HTML (check the file type indicator in the status bar of your editor), you can have Atom type out this entire skeleton for you. Just type the letters `html` followed immediately by the `[TAB]` key.

If the file type is something else, like Plain Text, it could be that you haven't saved your file yet so Atom doesn't know yet what kind of code you're writing. You can simply click the indicator to bring up a (very) long list of programming languages so you can select "HTML".


file type
indicator
LF
UTF-8
HTML
master
11 files

Lists

The two most common types of lists are *ordered* and *unordered*.

```
<ol>
  <li>Apples</li>
  <li>Bananas</li>
  <li>Cherries</li>
</ol>
```

and

```
<ul>
  <li>Apples</li>
  <li>Bananas</li>
  <li>Cherries</li>
</ul>
```

CODE READABILITY

Starting each element on its own line and indenting child elements makes the code much easer to scan at a glance, and also makes it easier to spot missing closing tags.

Would you rather read this:

```
<body><h1>
Hello! Here are some of my favorite fruits:
</h1>
<ol><li>Apples</li> <li>Bananas</li>
<li>Cherries</li></ol>
</body>
```

or this:

```
<body>
  <h1>Hello! Here are some of my favorite fruits:</h1>
  <ol>
    <li>Apples</li>
    <li>Bananas</li>
    <li>Cherries</li>
  </ol>
</body>
```

I thought so. :-)

Links

The web wouldn't be the web without links, originally called *anchors*:

```
<a href="https://www.amazon.com">Shop Now</a>
```

A link must have an `href` attribute, content, and a closing tag.

Images

The web would be a lot less fun without *images* such as photos and animated gifs:

```
<img
src="https://www.google.com/images/branding/googlelogo/2x/googlelog
 alt="Google Logo">
```

```
<img src="https://www.google.com/logos/doodles/2015/googles-new-
logo-5078286822539264.3-hp2x.gif" alt="Animated Google Logo">
```

- The `src` attribute is required.
- A closing tag is *not* required, because an image tag cannot contain any content.
- Other attributes are optional but often helpful:
    - `alt` provides alternate descriptive text
    - `width` and `height` will modify the image size

Tables

Tables are helpful for displaying numerical or tabular data sets.

Tables typically have a *head* and a *body*. The head consists of *rows* `tr` (usually just one) and *header columns* `th` . The body consists of rows and *table data* `td` .

City	Average Temperature
Chicago	35
Honolulu	75
Paris	69

Tables look terrible without some visual styling. We've applied some styling to the example table.

```
<table>
  <thead>
    <tr>
      <th>City</th>
      <th>Average Temperature</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>Chicago</td>
      <td>35</td>
    </tr>
    <tr>
      <td>Honolulu</td>
      <td>75</td>
    </tr>
    <tr>
      <td>Paris</td>
      <td>69</td>
    </tr>
  </tbody>
</table>
```

Forms

A *form* allows the user to input data. A form is a series of one or more interactive elements that allow the user to enter text or make selections.

Once the user has made their selections, the user *submits* the form to a server, which can process the incoming data. Therefore, although HTML can present a form, HTML alone cannot do anything with the data provided.

The `<form>` element encompasses all interactive elements. Each interactive element must have a *name* attribute so that the corresponding data can be distinguished from the other form elements.

See W3Schools for details on all of the possibilities.

We will highlight the `input` and `select` elements here.

`<INPUT>`

This is the most common form element and quite versatile.

The `type` attribute determines the behavior of the element.

Here are a few examples:

☐ Check Me

☑ Initially Checked

cookie@example.com

•••••••

○ Apples ○ Bananas ○ Cherries

What's your favorite color?

```
<p><input type="checkbox">Check Me</p>
<p><input type="checkbox" checked>Initially Checked</p>
<p><input type="text" placeholder="Enter some text..."></p>
<p><input type="password" placeholder="Say the secret word"></p>
<p>
  <input type="radio" name="fruits">Apples</input>
  <input type="radio" name="fruits">Bananas</input>
  <input type="radio" name="fruits">Cherries</input>
</p>
<p>What's your favorite color? <input type="color"></p>
```

SELECT LISTS

The `select` element uses child `option` elements to generate a list that the user can select from.

Choose one: Apples

```
Choose one:
<select>
  <option>Apples</option>
  <option>Bananas</option>
  <option>Cherries</option>
  <option>Figs</option>
  <option>Kiwi</option>
  <option>Oranges</option>
</select>
```

Visual Styles with CSS

Unlike HTML, CSS is a language that allows us to apply visual styles to our document.

CSS stands for *cascading style sheets*. CSS code is a list of style *rules*. Rules can be provided within an HTML `<style>` element, or they can live in a dedicated `.css` file.

> HTML is for marking up content; CSS is for styling.

For this course, all of our CSS rules will live in their own file, apart from the HTML content that they will be applied to.
We can apply a CSS file to any given HTML file by including an HTML line like this:

```
<link rel="stylesheet" href="styles.css">
```

where `styles.css` can actually be any filename you want.

The remainder of this section will demonstrate how to write CSS rules. We will assume that you will put them in a file named `styles.css` in the same directory as your HTML file.

Copy and paste this code into you HTML file and open it in your browser so that you can code along.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>My Page</title>
    <link rel="stylesheet" href="styles.css">
  </head>
  <body>
    <h1>Hello!</h1>

    <p>This is my amazing website.  Look around and
       find <span class="highlighter">amazing things!</span> This
is the main paragraph
       of this page.  Isn't it amazing?
     </p>

     <p>My favorite things are:</p>
     <ul>
       <li>Baseball</li>
       <li class="highlighter">Cookies</li>
       <li>Computer Progamming</li>
       <li>The audio archives of the Supreme Court</li>
     </ul>

  </body>
</html>
```

Colors

In your `styles.css` file, add the following line of code:

```
body {
  color: red;
}
```

Don't see any change? Make sure your `<link>` element is pointing to the correct `styles.css` file.

Save the file, and refresh your HTML file. All of the text should now be red!

MULTIPLE PROPERTIES

Change the selector to `h1` and add several properties:

```
h1 {
  color: red;
  border-bottom: 4px solid green;
  padding-bottom: 6px;
}
```

Play around! Change the colors and sizes in this rule to discover how they work.

Refresh your browser again, and notice the difference. Congratulations, your `h1` element is now the ugliest heading on the internet.

Hello!

This is my amazing website. Look around and find amazing things! This is t

My favorite things are:

- Baseball
- Cookies
- Computer Progamming
- The audio archives of the Supreme Court

RULE DEFINITION

A rule consists of a *selector* followed by a series of *properties*. Each property must end with a semicolon.


```
h1 {
  color: red;
  border-bottom: 4px solid green;
  padding-bottom: 6px;
}
```

#

Fonts

Fonts are specified with the `font-family`, `font-size`, and `font-weight` properties. Well, there are many others, but that's enough to get started. (For more properties, Google is your friend).

```
body {
  font-family: 'Arial';
  font-size: '20px';
}

li {
  font-weight: bold;
}
```

Notice how the `li` items will inherit the font characteristics of the parent HTML elements.

Borders

You can place borders around elements:

```
p {
  border: solid 3px green;
}
```

You can also control the amount of whitespace on each side of the border line. Use `padding` to give a little breathing space between the border and the interior content:

```
p {
  border: solid 3px green;
  padding: 6px;
}
```

Use `margin` for the outer spacing. You can also selectively apply margin and padding to the top, bottom, left, or right sides individually:

```
p {
  border: solid 3px green;
  padding: 6px;
  margin-left: 40px;
}
```

Hello!

This is my amazing website. Look around and find amazing things!

My favorite things are: #

- Baseball
- Cookies
- Computer Progamming
- The audio archives of the Supreme Court

Classes

We now come to one of the most powerful uses of CSS, which is to name a rule as a *classification* or *class*. A class is a set of CSS properties that can be applied to any HTML element at any time.

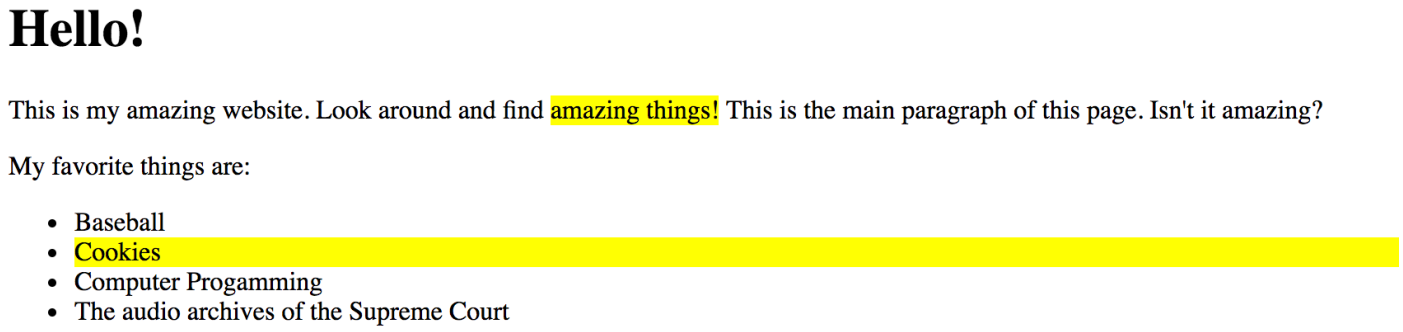
For example, suppose we decide to we often want to apply a yellow highlighter effect on certain elements, and we can't predict in advance which elements those will be.

We can define our class like this:

```
.highlighter {
  background-color: yellow;
}
```

Every HTML element that contains a `class` attribute of `highlighter` will have this rule applied to it, as in the example HTML at the beginning of this section.

Go ahead and refresh your browser to see the difference.



If you're wondering why the highlight extends all the way to the edge of the list element, we'll cover that in the next section.

Block Elements

Elements are classified into two broad groups: *block* elements and *inline* elements.

When the browser is deciding how to draw an element on the screen, the browser follows a certain set of rules referred to as *normal flow*, which basically means:

- Elements start at the top of the page and work down
- Elements start at the left and go toward the right

How the browser decides exactly which element goes where depends primarily on whether it's drawing a block or inline element at that moment.

BLOCK ELEMENTS START ON A NEW LINE

That's not exactly the technical definition but it's a great way to begin to think about it. Try this code inside of a `body` element:

```
Hello
<p>World!</p>
Goodbye.
```

View this code in your browser.

Why was "World!" drawn below "Hello"? The `p` element is a block-level element, so:

- It will start on a new line
- It will start on the left
- The height is determined by the font size (plus any padding and margin)
- It will reserve the full width on the screen for itself

That last point is why "Goodbye." appears on the line below. Even though there appears to be enough room to the right of "World", block-level elements reserve the entire horizontal space for themselves.

Common block elements:

- `h1` thru `h6`

- p
- div
- li
- form

Inline Elements

Inline elements are drawn "next to" the previous element.

```
<p>Hello <em>World!</em> Goodbye.</p>
```

Here, "World!" is drawn alongside "Hello" becase `em` is an inline element.

Inline elements only take up the minimum width necessary based on their content (plus any padding).

Common inline elements:

- `a`
- `img`
- `span`
- `em`
- `strong`

Grouping with `<DIV>`

Sometimes we need to group a bunch of elements together as a unit for purposes of CSS placement or styling.

A `div` has no visible representation itself, but is simply a grouping construct so that you can apply a CSS rule to the group as a whole.

THE PROBLEM

Consider this problem: we'd like to draw a single green border that surrounds all three paragraphs below.

```
<p>I like to eat cookies!</p>
<p>Baseball season starts in April.</p>
<p>Chicago is cold in the winter.</p>
```

I like to eat cookies!

Baseball season starts in April.

Chicago is cold in the winter.

The following CSS doesn't quite work. It draws three borders instead of one (can you explain why?):

```
p { border: solid 3px green; padding: 12px; }
```

I like to eat cookies!

Baseball season starts in April.

Chicago is cold in the winter.

THE SOLUTION

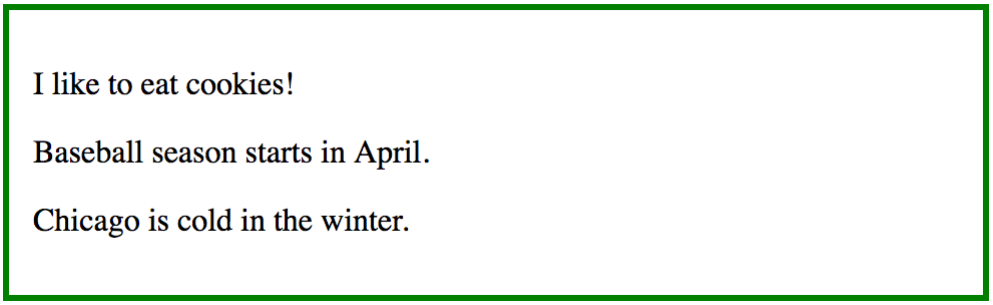
First, we group the elements together with a `div` and apply a class for our CSS rule:

```
<div class="framed">
  <p>I like to eat cookies!</p>
  <p>Baseball season starts in April.</p>
  <p>Chicago is cold in the winter.</p>
</div>
```

Next, we rewrite our CSS rule as a class definition:

```
.framed { border: solid 3px green; padding: 12px; }
```

Presto!



Grouping with

``

A `span` is simply the inline equivalent of `div`. Sometimes we need to group a bunch of inline elements together as a unit for purposes of CSS styling.

A `span` has no visible representation itself, but is simply a grouping construct so that you can apply a CSS rule to the group as a whole.

THE PROBLEM

Consider this problem: we'd like to draw a single green border that surrounds the phrase "cold in the winter:"

```
<p>Chicago is cold in the winter.</p>
```

The following CSS doesn't quite work. It draws a border around the entire paragraph (can you explain why?):

```
p { border: solid 3px green; padding: 12px; }
```

THE SOLUTION

First, we group the elements together with a `span` and apply a class for our CSS rule:

```
<p>Chicago is <span class="framed">cold in the winter.</span></p>
```

Next, we rewrite our CSS rule as a class definition:

```
.framed { border: solid 3px green; padding: 12px; }
```

Try it!