HyperText Markup Language and Cascading Style Sheets

Introduction

HyperText Markup Language or HTML is the language that is probably the most commonly associated with the web. HTML is responsible for organizing the content of a web page. This page has things like headings and under headings is usually a section. Then some things are defined as links or there might be lists. All of this is defined in HTML. Here is a sample of what is a minimal HTML page.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>title</title>

</head>

<body>

<!-- page content -->

</body>

</html>

This particular example page does not actually have any content, just the minimal amount of meta-data to be considered a valid page. The actual content would go within the body.

Notably I never mentioned HTML affecting how a page looks. Browsers have some default ways of rendering things, but HTML is a language to organize data. It is not a language to lay it out or to change the way it looks. That is what Cascading Style Sheets (CSS) are for. These files apply certain styles like a color or positioning rule to matched elements. The following is a CSS snippet that applies some rules to paragraphs and to things in the 'lead' class.

p {

margin: 0 0 9px;

font-family: "Helvetica Neue", Helvetica, Arial, sans-serif;

font-size: 13px;

line-height: 18px;

}

p small {

font-size: 11px;

color: #999999;

}

.lead {

margin-bottom: 18px;

font-size: 20px;

font-weight: 200;

line-height: 27px;

}

By the time you are done with this section you should easily be able to parse these rules and write your own.

Key Questions

* What is the purpose of HTML?
* What is the purpose of CSS?
* When would you use CSS vs HTML?
* Where should classes and ids be used?
* What are the CSS selectors and how are they used?
* What are the various components of an HTML tag?

Assignment Overview

This week you will be making a non-interactive website. It will have constraints on the way the data should be organized and styled. You will need to appropriately use HTML and CSS to make a website that meets those constraints.

Explore the Topics

* [Intro to HTML - Organizing the data of the world](http://eecs.oregonstate.edu/ecampus-video/CS290/core-content/html-css/html-intro.html)
* [HTML Tags - Putting things in the page](http://eecs.oregonstate.edu/ecampus-video/CS290/core-content/html-css/html-tags.html)
* [A Smattering of Tags - Some tags you might often use](http://eecs.oregonstate.edu/ecampus-video/CS290/core-content/html-css/html-tag-examples.html)
* [Forms - A subset of really important tags](http://eecs.oregonstate.edu/ecampus-video/CS290/core-content/html-css/html-forms.html)
* [Intro to Cascading Style Sheets - Making things pretty!](http://eecs.oregonstate.edu/ecampus-video/CS290/core-content/html-css/css-intro.html)
* [CSS Properties - Make things look like different things](http://eecs.oregonstate.edu/ecampus-video/CS290/core-content/html-css/css-properties.html)
* [CSS Selectors - Make those things in particular look like different things](http://eecs.oregonstate.edu/ecampus-video/CS290/core-content/html-css/css-selectors.html)

Additional Resources

* [Mozilla's HTML Intro](https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Introduction) - This is a very minimal introduction to the concepts behind HTML. It does not go over specific tags but is some good material to have as background as you go into the class content. (Required)
* [Mozilla's CSS Intro](https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Getting_started) - This is an excellent getting started guide for CSS. It is 13 sections but they are not much longer than a single page each. Ignore section 14 and don't worry about pseudo-classes in section 5. (Required)
* Mozilla's [HTML](https://developer.mozilla.org/en-US/docs/Web/HTML) and [CSS](https://developer.mozilla.org/en-US/docs/Web/CSS) resources. There are some good getting started guides here but I recommend these sites more as a less technical reference. (Reference Material)
* W3C [official HTML specification](http://www.w3.org/TR/html5/). This is very technical, but less so than the spec on a language like C++. If you want to know the exact definition, look no further. I would not recommend this as a general reference however. (Reference Material)
* Jon Duckett's HTML and CSS: Design and Build Websites (ISBN:1118871642). This book is in no way required, but it is a very visual HTML and CSS book which exists in the physical world. (Optional Book)

Reflection

There is a lot of material this week. I don't anticipate that you will remember it all going into upcoming weeks. But you should have a basic understanding such that you can have these pages open and get a basic web page created and styled. You should also feel comfortable using some sort of reference to look up HTML or CSS elements. Finally you should start getting used to using Chrome Developer Tools to examine other peoples websites and to help debug your own. There will be a bit of a break moving forward from here where we focus on pure JavaScript, but then you will be getting back to a lot of HTML and CSS work as we start modifying pages using JavaScript.

# Intro to HTML

## Introduction

HTML, standing for HyperText Markup Language, is the language used throughout the web to define the way information is organized. Breaking this down, HyperText refers to documents that link to one another. This is one of the defining attributes of HTML is that one can navigate from one document to another. The other piece is that it is a Markup Language. This means that it is basically a language to add meta-data to existing data. This is distinct from a programming language which allows for the use of logic and control structures.

## The Purpose of HTML

### **What is it for?**

HTML is used to organize, describe and give meaning to data. That's it.

### **What is it NOT used for?**

This is where things get much more complicated. For quite some time HTML was used as a layout tool. In the early years of the web there was little other option. If you wanted any sort of complicated layout it was almost always achieved by used a table to do things like create multi column layouts. This seems to make sense, tables have columns and rows. If I want my page to have columns and rows, I can just put it into a table!

But wait, does formatting my blog about kitten pictures into two columns using HTML actually give it any additional meaning? Is it really organizing the data?

Tables ought to hold tabular data. That is rows and columns contain somehow related data and being in a particular row or column should have some meaning. A spreadsheet might have a column for income and another for expenses and a row giving totals at the bottom. These things give meaning to the data in the rows or columns. We know if a number is in the expense row it was an expense.

Breaking a list of images into two columns to make it more visually appealing does not add additional meaning, so this is not a task for HTML. Rather it will be a task for CSS which we will talk about in an upcoming section.

## Versions of HTML

This is becoming a less troublesome topic. HTML recently made the switch from HTML 4.01 to HTML5 and HTML5 is pretty much universally adopted by browsers these days. If you get contracted to update some internal web application for a company you may run into HTML 4.01 but in general, we will be working with HTML 5 which is adds a lot of additional elements to help give additional meaning to data and is better suited for the most common types of content on the web. If you look up reference material, make sure it is for HTML5 because there is still a lot of highly ranked information on HTML 4.01 on popular search engines.

## Activity

No activities yet.

# HTML Tags

## Intro

HTML structures documents by using elements. Elements can stand on their own or be nested within other elements. For example a paragraph element might exist within a section element or a table row will be within a table. Everything in HTML is an element. The way elements are represented in HTML is by using tags. Here is an example of a tag.

<section class="example">Hello World!</section>

Below we will talk about the various pieces of this tag, what else could be added to it and what could be removed from it.

## Anatomy of a Tag

Lets look again at tag from the introduction and talk about the various pieces.

<section class="example">Hello World!</section>

Every single tag is going to open with a < symbol followed by the name of the tag. In this case it is a section tag. Some tags like a line break <br> consist only of the opening and closing angle bracket and the name. But most tags also require a closing tag. This is a tag that opens with </ instead of <. So in the above example the closing tag is </section>. Everything between the opening tag and the closing tag is the content of the tag. In this case the content is Hello World!.

In addition to basic opening and closing tags additional information can be provided using attributes. In the above example the section belongs to the class "example". This is usually specified using the syntax of attribute-name="attribute-value" or attribute-name="attribute-value" on occasion you will see the quotation marks omitted or an attribute name with no associated value. For example, a check-box is checked if the attribute checked exists and it is unchecked if that attribute does not exist. The value of the checked attribute is meaningless.

For this class if an attribute value exists it should be wrapped in quotes. You can choose either single or double quotes but be consistent throughout your HTML. Always use single quotes or always use double quotes.

## Nesting Elements

Opening and closing tags act a lot like parenthesis. If a tag has both an opening and closing tag those must both appear at the right depth inside the tree of elements. For example, you could have two paragraphs, one of which has emphasized text, inside a section like this:

<section>

<p>I am paragraph 1.</p>

<p>I <em>am</em> paragraph 2.</p>

</section>

As it turns out, in certain circumstances, the ending tag of a paragraph can be omitted so this would also be valid:

<section>

<p>I am paragraph 1.

<p>I <em>am</em> paragraph 2.

</section>

However the following would not be valid because the emphasized text <em> opens inside of the paragraph but closes outside of it:

<section>

<p>I am paragraph 1.</p>

<p>I <em>am paragraph 2.</p></em>

</section>

Many text editors will highlight issues with mismatched tags. There are also on-line [validators](http://validator.w3.org/) which can scan your HTML document and find errors like this.

## Activity

The following code sample has some issues. You should fix the tags so that it is valid. In addition to fixing the tags, you should give the inner section a class attribute with the value of content. In addition, we can optionally replace the quotation marks with opening and closing q tags because they represent actual quotes. Finally fix the emphasis tag (em) tag so that it only emphasizes that the business was in danger.

[Edit in JSFiddle](https://jsfiddle.net/wolfordj/Lbqoczzs/1/?utm_source=website&utm_medium=embed&utm_campaign=Lbqoczzs)

<section>

<h1>Tags Exercise</h1>

<section>

<p>Milk production at a dairy farm was low and <em>the business was in danger. The farmer had hired vets, biologists, chemists all with no success. As a final resort the farmer hired a physicist to help him solve the problem. After days of work the physicist told the farmer "I have a solution!".

<p class="dialogue">"The farmer replied "Great what is it?"

<p class="dialogue">"First we must assume spherical cows on an infinite frictionless plane..."</p></p>

</section>

</section>

|  |
| --- |
| **Result** Tags Exercise Milk production at a dairy farm was low and the business was in danger. The farmer had hired vets, biologists, chemists all with no success. As a final resort the farmer hired a physicist to help him solve the problem. After days of work the physicist told the farmer "I have a solution!".  "The farmer replied "Great what is it?"  "First we must assume spherical cows on an infinite frictionless plane..." |

# [Edit in JSFiddle](https://jsfiddle.net/wolfordj/s7w2kLps/5/?utm_source=website&utm_medium=embed&utm_campaign=s7w2kLps)

## Solution

Switch to HTML view to see the solution code.

# Tags Exercise

Milk production at a dairy farm was low and the business was in danger. The farmer had hired vets, biologists, chemists all with no success. As a final resort the farmer hired a physicist to help him solve the problem. After days of work the physicist told the farmer I have a solution!.

The farmer replied "Great what is it?

First we must assume spherical cows on an infinite frictionless plane...

<section>

<h1>Tags Exercise</h1>

<section class="content">

<p>Milk production at a dairy farm was low and <em>the business was in danger</em>. The farmer had hired vets, biologists, chemists all with no success. As a final resort the farmer hired a physicist to help him solve the problem. After days of work the physicist told the farmer <q>I have a solution!</q>.

<p class="dialogue"><q>The farmer replied "Great what is it?</q>

<p class="dialogue"><q>First we must assume spherical cows on an infinite frictionless plane...</q>

</section>

</section>

## Review

This should get your familiar with the syntax and various pieces of HTML tags. We still don't know what specific tags do or why we want to give them things like IDs or classes, but that is coming very soon!

# HTML Tag Examples

## Intro

In this module we are going to look at a small subset of tags in detail so we can look at how they are actually used. Then we will look at a larger collection of tags and talk about how they what they are used for in more general terms.

## Headers (h1-h6)

Headers are a pretty intuitive tag. Things within a header tag are represented as a header. This usually means they are given a larger font and sometimes a bold font or even a different typeface. A header tag is created like so:<h1>An Important Header</h1>. The opening and closing tags are always required. h1 is the most important or highest level header. Headers run from h1 through h6 with h6 being the lowest level header that you might find deep in a paragraph. In the document you are reading right now the title at the top of the page is an h1 and the header at the top of this section is an h2. If there are subsections they are h3 headers.

By default headers are block level elements. This means they get their section of the page. If a page break does not exist before or after it one will be created to ensure it is on its own line.

### **When to Use Headers**

If you would potentially find something listed in the table of contents it is a great candidate for a header. Headers should be used to add organization to a document. So any section or sub section that needs a title can use a header for it. One could make a good argument that they could also be used to add a title to something like an image. But it would not be good to use as the title for a table because tables have their own element for adding a title called caption.

### **When not to Use Headers**

Headers make text big and obvious. It might be tempting to use something like an h5 or h6 to add emphasis to a word in a paragraph or maybe use it as a way to highlight dialog. This is not what headers are to be used for. It might look good, but things like screen readers are going to use headers to section content for visually impaired users. It will really confuse those tools if headers are used to just add emphasis to text or just to style it.

## Section, Article and Div

These elements along with a few others are used to section off content. Visually these do very little on their own. They will usually break their content into a block so that a line break comes before and after. Beyond that how they are styled really depends on the CSS that is used. Although they often all do the same thing visually it is important to understand why they are used.

### **The Div**

The div element <div>Content Here</div> is used to divide content. It should be used as a last resort when no other element makes sense. If you want to divide content purely for stylistic reasons this is the element to use. It conveys no meaning and just generically divisions content.

### **The Section**

The section element:

<section>

<h2>Some Section</h2>

<p>I am content in the section</p>

</section>

is used to make a thematic grouping of content. In this document I have "Section, Article and Div" as a section and "Headers" as a section. It is a group of content that is all related but does not quite stand on its own. Usually a sections first child will be a header that describes what is in that section. If there is not a good way to classify the content in the section using a header, you may want to consider using a div instead. A section should only be used if all the content is related.

### **The Article**

The article element is structurally the same as the section except that it is an <article> tag instead of a <section> tag. The requirements for a article are stricter than a section. Not only should the content all be related but the content should generally stand on its own as a composition.

For example, this section of the page just describing the article element makes references to other parts of the page. So it is all thematically grouped around the article element. But it is not a complete composition. So it is left as a section rather than made into an article. This page as a whole could be an article as it is a complete composition describing element examples.

## Anchors (a) and Images (img)

### **Anchors**

The anchor <a href="http://foo.com">A link to foo</a>. is how one links from one page to another. Both the opening and closing tag are required. Anything between the tags will represent the link someone can click on to navigate to the target specified by the anchor. The href attribute is what is used to specify where the link will take the user when it is clicked.

### **Images**

The img tag is used to display images: <img src="kitten.png" alt="A Cute Kitten">. The src attribute indicates where the image is located. No closing tag is used for images. By default images display in-line. So no new line is made for them. We will talk a bit more about specify the source for images later on as identifying the location of resources can get a bit tricky at times.

## Strong, B, Em, i...

This family of elements is for adding meaning or style to specific text within a paragraph. This is different from the previous elements which generally served to group text.

### **Strong**

The strong element: <strong>I am important</strong> marks text **that is more important**. It requires and opening and closing tag. The important thing about strong is that it signifies that text is important. Usually browsers will make it bold by default. But it could make it red, it could make it underlined, it could make it blink (don't do this...). By marking text with strong you are saying it is more important that the text around it.

### **B**

The b element makes text stylistically different from other text. Look at me, I am <b>different</b>. This could be used to highlight **keywords** in a paragraph or conform to style guidelines set by some journal. It does not give additional meaning to the text. Text inside a b element is no more important than text outside of the b element. It just is styled differently.

### **Em**

The em element adds emphasis to a word. Often making it italic. "I love kittens" suggests that the emphasis is on the I. So maybe someone is differentiating themselves in a room of dog lovers. On the other hand "I love kittens" suggests that it is only kittens that I love and that I may not be such a big fan of older cats. It is adding additional meaning to the word or phrase that is emphasized. There is also an i element which will make text italic but not convey additional meaning. Again this might be used to conform to style guides or the like but should not be used when wanting to add emphasis to a word.

## Lists and Tables

So far we have looked at elements that really were self contained or could include all kinds of different stuff. But there are elements that are more complex. Lists and Tables fit into this category as they are really collections of several nested elements.

### **Lists**

Lists come in several different flavors, ordered, unordered or definition list. We will used an unordered list in our example here:

<ul>

<li>first item</li>

<li>second item</li>

<li>third item</li>

</ul>

The ul element starts and ends the unordered list. It can only contain list items (li) as children. Those list items can however contain most other HTML elements. The above list will looks like this:

* first item
* second item
* third item

Often times styling is used to add or remove bullet points to the list. Ordered lists will be numbered or otherwise sequenced, unordered lists will not. Here is the same list, now ordered (ol instead of ul):

1. first item
2. second item
3. third item

Definition lists (dl) are similar but have pairs of terms and definitions

<!-- Code example from https://developer.mozilla.org -->

<dl>

<dt>Name</dt>

<dd>Godzilla</dd>

<dt>Born</dt>

<dd>1952</dd>

<dt>Birthplace</dt>

<dd>Japan</dd>

<dt>Color</dt>

<dd>Green</dd>

</dl>

This is rendered as:

**Name**

Godzilla

**Born**

1952

**Birthplace**

Japan

**Color**

Green

### **The Table**

To finish this off we will look at what is probably the most structurally complex HTML element out there, the table. Here is an example table we will look at. Below is a sample table from the w3c standard.

<table>

<caption>Characteristics with positive and negative sides</caption>

<thead>

<tr>

<th> Characteristic

<th> Negative

<th> Positive

<tbody>

<tr>

<th> Mood

<td> Sad

<td> Happy

<tr>

<th> Grade

<td> Failing

<td> Passing

</table>

And here is what that table looks like when it is rendered in your browser:

| Characteristics with positive and negative sides | | |
| --- | --- | --- |
| **Characteristic** | **Negative** | **Positive** |
| **Mood** | Sad | Happy |
| **Grade** | Failing | Passing |

So lets use a definition list here to talk about what all these pieces are doing.

**Table**

This opens and closes the table, it is pretty straight forward. Everything related to the table will be inside these tags.

**Caption**

This represents the title and description of the table. It can include just the title or additional explanation of the table.

**thead**

This is the table header. It can contain rows that represent the header of the table. So these would be rows that do not contain actual data but instead contain things like labels signifying the data that will be in those columns. The rows in this section would typically contain th elements.

**tbody**

This is the body of the table and contains the rows that actually have data in them rather than the descriptive rows that would be in the header. The rows in this section would typically contain td elements.

**tr**

The is a row of the table. It contains either td elements (which are data cells) or th elements (which are header cells).

**td**

This is a data cell, it should contain data from the table. Each cell in a row will typically signify a column if every row has the same number of cells. If you want to have a cell span multiple columns you can you use the colspan attribute.

**th**

This is a header cell it should signify what is in the column below it or the row beside it.

## A Sample Skeleton

Beyond the above tags there are some basic tags that should exist in every page. Below is a skeleton you can use to get started when making an HTML page.

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>title</title>

<link rel="stylesheet" href="stylesheet.css" type="text/css">

</head>

<body>

</body>

</html>

The content in the head is metadata. It will help the browser and robots understand what is on the page. The link tag is where we can link to a file that contains CSS rules to style the page. The actual content the user will see all goes in the body of the page.

## Activity

As an activity you should try to make a moderately complex table. To do this you will certainly want a reference to look up all the options. I strongly recommend [Mozilla's table content reference](https://developer.mozilla.org/en-US/docs/Web/HTML/Element#Table_content). Your task is to as close as possible, recreate the table shown below.

[Edit in JSFiddle](https://jsfiddle.net/wolfordj/0rgxgnos/4/?utm_source=website&utm_medium=embed&utm_campaign=0rgxgnos)

| **Column 1 heading** | **Column 2 heading** | **Column 3 heading** |
| --- | --- | --- |
| Row 2, cell 1 | **Row 2, cell 2, also spanning Row 2, cell 3** | |
| Row 3, cell 1, also spanning Row 4, cell 1 | Row 3, cell 2 | Row 3, cell 3 |
| Row 4, cell 2 | Row 4, cell 3 |

Go to [this fiddle](http://jsfiddle.net/wolfordj/s6apw402/) at jsfiddle.com and using just the HTML panel put in HTML that will create the example table. CSS is added to create the borders so it is easier to see what is going on. If you get stuck you can see how it was made below by clicking on the HTML panel

| **Column 1 heading** | **Column 2 heading** | **Column 3 heading** |
| --- | --- | --- |
| Row 2, cell 1 | **Row 2, cell 2, also spanning Row 2, cell 3** | |
| Row 3, cell 1, also spanning Row 4, cell 1 | Row 3, cell 2 | Row 3, cell 3 |
| Row 4, cell 2 | Row 4, cell 3 |

<table>

<thead>

<tr>

<th>Column 1 heading</th>

<th>Column 2 heading</th>

<th>Column 3 heading</th>

</tr>

</thead>

<tbody>

<tr>

<td>Row 2, cell 1</td>

<td colspan="2"><b>Row 2, cell 2, also spanning Row 2, cell 3</b></td>

</tr>

<tr>

<td rowspan="2">Row 3, cell 1, also spanning Row 4, cell 1</td>

<td>Row 3, cell 2</td>

<td>Row 3, cell 3</td>

</tr>

<tr>

<td>Row 4, cell 2</td>

<td>Row 4, cell 3</td>

</tr>

</tbody>

</table>

## Review

This should give you a good understanding of the basic HTML elements. There are lots of other elements but with these you should be able to put together a complete page and it should give you an idea of the different categories of elements you might find in HTML.

# HTML Forms

## Intro

In this section we will discuss HTML forms and some of the behind the scenes workings of them. Pretty much any time you are submitting data via a website you are using a form. If you are creating a new account at Amazon, logging into OSU's website or uploading a file, that is all happening with forms. So it is important to get a good grasp of what is going on on the HTML side of things to submit data before we talk about actually processing that data on the server. Below is an example form that we will use to discuss the various parts of a form.

<form action="http://www.example.com/review.php" method="get">

<fieldset>

<legend>Your Details:</legend>

<label>Name:

<input type="text" name="name" size="30" maxlength="100">

</label>

<br />

<label>Email:

<input type="email" name="email" size="30" maxlength="100">

</label>

<br />

</fieldset>

<br />

<fieldset>

<legend>Your Review:</legend>

<p>

<label for="hear-about">How did you hear about us?</label>

<select name="referrer" id="hear-about">

<option value="google">Google</option>

<option value="friend">Friend</option>

<option value="advert">Advert</option>

<option value="other">Other</option>

</select>

</p>

<p>Would you visit again?

<br />

<label>

<input type="radio" name="rating" value="yes" />Yes</label>

<label>

<input type="radio" name="rating" value="no" />No</label>

<label>

<input type="radio" name="rating" value="maybe" />Maybe</label>

</p>

<p>

<label for="comments">Comments:</label>

<br />

<textarea rows="4" cols="40" id="comments"></textarea>

</p>

<label>

<input type="checkbox" name="subscribe" checked="checked" />Sign me up for email updates</label>

<br />

<input type="submit" value="Submit review" />

</fieldset>

</form>

## The Form Tag

When dealing with forms everything will live within a form tag. This tag must have an opening and closing tag. In general any content can show up inside a form tag except other forms. In addition to general content like paragraphs, headers or images forms almost always include input elements. These elements are where the user will actually enter in or select options to submit. There are other things like a textarea which function in much the same way as an input element but are just called something different.

If you look at the HTML of the form above you will see at the outermost level is a form tag, then within that there are many input elements and other elements like fieldset or label to help organize the form.

### **Form Attributes**

The form tag itself has two very important attributes. The first is the action. The action attribute specifies where the form should be sent. It is similar to an address on a piece of mail. When the form is submitted the browser will package up all the contents of the form and send it to the location specified in the action. The second important attribute is the method. This specified how the data will be sent. There are two primary ways a web browser will submit a form one is a get the other is a post. They both will send the same data but it will be handled differently on both the browser and the server.

## Inputs

### **The Input Tag**

The input tag is the tag most commonly used to gather data within a form. It stands out a little from other elements because it has an attribute type which vastly changes how it is displayed, how it is used and even what other attribute tags can be used. In addition to have this type attribute it is also critical that it have a name attribute. The name is how the server will know which part of the form the data is associated with. For example, in the above form the email field has the attribute name="email" it happens to also match the value given to the type attribute but this is just a coincidence. If one were to enter the email *foo@bar.com* and submit the form the server would receive something which looked like this email=foo@bar.comso it would know the field named email had the value *foo@bar.com* entered into it.

There are a lot of different kinds of inputs and attributes. I would recommend [Mozilla's Input Reference](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input) if you ever need an answer to Is there an input for this?. In addition it enumerates all the other possible attributes which can do things like limit length or do some types of validation.

#### **Radio Buttons and Check Boxes**

These inputs need their own special section as their use is not entirely intuitive. Check boxes need both a name and value just like other inputs. However the server will only ever know about the existence of the check box if it is checked. For example this check box <input type="checkbox" name="foo" value="bar"> will send the server foo=bar if it is checked. If it is unchecked the server will not get any indication that the check box even exists.

Radio buttons add on a little twist. If no radio button is selected, nothing will get sent, just like with a check box. But every radio button that has the same name will be in a radio button group. Within that radio button group only one radio button can be selected. In the example above for the Would you visit again? portion of the form those radio buttons are all in the same group. So you cannot select both yes and no at the same time. If they had different name attributes then they could all be selected at once.

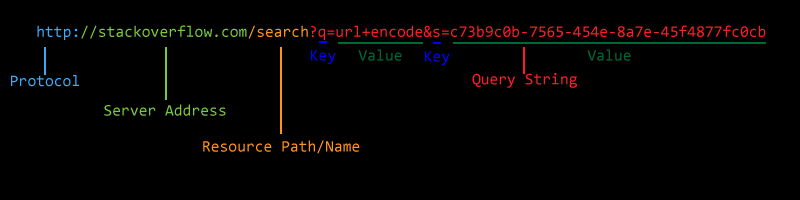
## GET and POST

GET and POST requests are the primary way for the browser to send data other than the data usually sent in the headers to a server. When submitting standard HTML forms these are the only options for sending the form data to the server. It is important to understand the differences and to know when one would be preferred over another.

Both GET and POST are going to send key value pairs. On the server side these values are often accessed via an associative array. The server will look up values by supplying a key. For example username=alice has the key username associated to the value alice. This course will often talk about the data sent from both GETs and POSTs assuming they are represented as an associative array.

### **GET**

A GET request sends the key value pairs as parts of the URL. This is the primary identifying attribute of a GET request sent from a server. Here is an example URL of a search on StackOverflow which is submitted via a GET request http://stackoverflow.com/search?q=url+encode&s=c73b9c0b-7565-454e-8a7e-45f4877fc0cb. We will use this as an example to look at the syntax of a GET request.



Lets break this down piece by piece http: is the protocol. It tells us this is using Hyper Text Transfer Protocol. Often times you will see https: which is the secure version of the protocol. Next is //stackoverflow.com. This is the address of the server we are trying to contact. After that will come the path to the resource we are requesting. In this case we are only one directory deep in /search and the server is able to interpret that as a resource. Other times you will see a directory chain ending in a file like /foo/bar/kitten.jpeg. Now what comes next is how we can tell this is probably a GET request. The ? indicates that it is the start of a query string. In that query string there is going to be a specific format for key value pairs. That format is key1=value1&key2=value2&keyn=valuen. So keys are assigned values using = and key value pairs are separated using & symbols. So in this instance the q key has the value url+encode and the s key has the value c73b.... In this case q probably stands for query and s for session but there is no real convention here.

### **POST**

A POST sends the same kinds of data as a GET, a list of key value pairs. What is different is that instead of sending it as part of the URL it is send as part of the request body. That means that you cannot tell what data (if any) was sent to a server via a POST request by just looking at the URL. It will still arrive at the server as part of a query string in the same format as a GET query string. So it will have key value pairs separated by &l; symbols.

### **GET vs POST**

Remember that the location of the query string, in the URL vs the request body is the main difference between a POST and a GET. With that in mind think about what the advantages and disadvantages of each might be.

## Activity

Create a simple form with a few inputs (I would encourage you to play around with some of the more complex types like check boxes and radio buttons). Set the forms action to http://classes.engr.oregonstate.edu/eecs/winter2015/cs290-400/tools/class-content/form\_tests/check\_request.php. When you submit the form this page will tell you what type it was (GET or POST) and also show you all the variables and their values.

## Review

The purpose of this module is to introduce you to forms. You should be comfortable creating a form to take common types of data and submit it to a website somewhere. Later we will be learning more about the specifics of how GET and POST are handled. For now you should be aware that GET values are sent in the URL and POST values are sent in the body of the request. In the future forms will be the main way you send user date to either your server or other servers.

# Introduction to CSS

## Intro

Cascading Style Sheets (CSS) are the layout tool used for specifying how a web site looks. When you go to a page and see a navigation bar, a footer and images embedded in a news article with text wrapping around them it was CSS that made that happen. This is a really critical thing to remember. CSS defines the look of a page whereas HTML structures and gives meaning to content.

## CSS Properties

The majority of CSS is about specifying properties and giving those properties values. There are well over a hundred different properties and far more possible values they can be assigned. Not all properties make sense on all elements.

An example property is color. The color property sets the text color of an element. For example This spans color was set to purple. The syntax for a property/value pair in css is property: value;. So to set the color to purple the code is color: purple;.

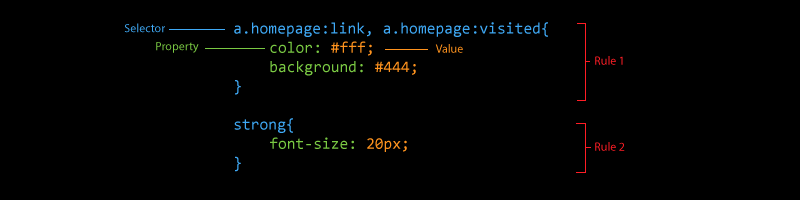
## CSS Selectors

Another critical piece of CSS is selectors. We do not want to have to set the font of every single paragraph every time we write a new paragraph. That would be a tremendous amount of work and near catastrophic if one ever decided they wanted to switch from a serif to sans-serif font.

Selectors let us set rules for which elements styles will apply to. They can be as simple as applying a style to everything using the \* selector or we can be as specific as applying a style to every other row of tables which exist in sections belonging to the class "data-display" section.data-display tr:nth-of-type(odd). In general you will not be using selectors as specific as this but you can if you need to.

## Syntax of CSS

CSS syntax has really only two parts, the selector syntax and the style syntax. Here is an example of a pair of CSS rules.



The selector comes before the curly braces. Within the curly braces are the CSS property and value pairs separated by semi-colons. Each CSS rule is usually separated by a new line for readability.

## Activity

Go to a few websites you visit frequently. Use Chrome developer tools to inspect some elements and look at the styles which are applied to them and look at the selectors they use to apply those styles.

## Review

This sub-module should give you a basic understanding of what CSS is and allow you to recognize CSS code when you see it. You may not know what the selector rule is actually doing or what the style is actually changing but at this point you should be able to look at a file and tell if it is an HTML file or a CSS file by looking at the code. In the next modules you will learn more about the common selectors and a few important CSS properties.

# Common CSS Properties

## Intro

As mentioned before there are a lot of CSS properties. Some vary from browser to browser. Some are only in the spec but are not widely supported. As a web developer you will need to get comfortable looking through lists of properties looking for one that might do what you want. The other thing you will want to do is when you see an interesting styling on a website, inspect it and see what properties they used to do it.

All of that said there are a few properties you are going to be using all the time and some of them are not entirely intuitive and those properties are what we will be discussing in this section.

## Colors

Colors in CSS are represented in a couple ways. One is hex values in the form of #RRGGBB. Larger R values give more red, G green and B blue. So #FF0000 would be pure red. #FFFFFF would be white. #550055 would be a darkish purple. In addition to using hex there are several named colors. A list of the basic named colors and their corresponding hex values can be found [here](http://www.w3.org/TR/css3-color/#html4). Personally when I am just using colors to help separate things for layout I use named colors. When actually designing a sites color scheme I use hex colors for more control.

### **Some Example Color Properties**

**color**

The color of the font in an element.

**border-color**

The color of the border of an element

**background-color**

The background color of an element

## Positioning

CSS is often used to position elements. When we talk about the position of an element we need to say what it is in relation to. In the real world we might say that a treasure is buried at 56 degrees N and 12 W. Or we might say it is 300 miles West of Glasgow. In the same way in CSS we can talk about where an item is relative to the screen or relative to its ancestor element.

### **Static**

Static positioning is the default positioning. The browser will figure out where something ought to be using the default algorithm. AS a designer you don't have a lot of control here.

### **Relative**

Objects with relative positioning are placed where they normally would be then the change in their location is measured from that spot.

### **Absolute**

This sets the position based on the parent element. If an elements absolute position is set to 0 by default the top left corner of the element be in the exact same position as the top left corner of its parent element.

### **Fixed**

This fixes the position of the element with respect to the document window. If you scroll the page the fixed element will scroll with the page. So you can make a nav bar always be at the top of the page. I would suggest not using fixed placement of elements as it can make for a difficult user experience on smaller screens.

## Measurements

We know how to position things but what about actually measuring distances? In CSS there are a lot of options for measurement. We are only going to look at two px and em.

px is an absolute measurement in pixels. We are going to assume a pixel is equal to one pixel on a display. Sometimes this isn't true. We are going to pretend it is and move on.

em is a relative measure based on the font size of a page. This is a nice tool to use for layouts because if a user increases their font size to make a site easier to read the size of elements will expand as well. If you were to use a fixed measurement and a user increased the font size text might spill outside of the elements which are supposed to contain it. Finally % can be used which is a measurement of the parent. So setting an element to width:50%; will make it half as wide as its parent element.

## Spacing

Finally we are going to talk about spacing. When laying out documents essentially everything is a box. Inside every box is the content of the element. When laying out these boxes we can specify two things how far one box is from another and how much space there is between the edge of a box and its content. We refer to these as margin and padding respectively.



From this position the sides left right top bottom are used to move the element. left: 5px; means that the left edge of the box will be offset away from its normal position moving the element 5px right. top: 10px; means the top of the element will be 10 pixels from where it normally is shifting the element down 5 pixels.

## Activity

Go to [this fiddle](http://jsfiddle.net/wolfordj/1nycqpj7/5/). In the upper right box you can modify the CSS. Play around by setting different position properties and setting different top or left values to move the divs around.

## Review

This sub-module is designed to help you learn the basics of positioning elements. There is a lot more to learn beyond what was introduced here but this should let you achieve almost any layout even if there are cleaner ways to do it with more advanced features.

# CSS Selectors

## Intro

We know about the basic CSS properties and how we can fiddle with the style of elements. Now we need a way to actually apply styles to specific elements and that is where selectors come in. Selectors, unsurprisingly, let us select element to apply styles to. Broadly speaking we can select things based on its type, its class or its id. When two selectors try to apply different values to the same property things targeting id have the highest precedence, then class then type.

## The ID Selector

The ID selector targets the single element with a matching ID and is of the following syntax #id. So to target a form with an id of "form1" we would use #form1. This is fairly straight forward as there should only ever be one element with that ID.

## The Class Selector

The class selector selects all elements of a given class. The syntax is .class. So .navigation will target all elements with the class attribute of 'navigation'.

## Type Selectors

Type selectors get significantly more complicated. These selectors select elements based on their type and relationship to other elements.

**Basic Type**

You can specify an element name directly. For example p will target all p elements.

**Descendant Selector**

This will target all elements of type 'x' which are a descendant of type 'y' with the syntax y x. So nav ul will apply a style to all ul elements which are contained in a nav element.

**Child Selector**

The child selector will target all element of type x which are direct children of type y. The syntax is y>x. So nav>ul would target ul elements who's direct parent is a nav element. But unlike the decedent selector it would not target a ul within a div within a nav because the ul is a child of the div, not the nav.

**Adjacent Sibling**

This will target all of element type x which is at the same level on the tree and immediately after an element of type y. The syntax is y+x. So h1+p would target a p if it directly followed an h1. You might use this if you want your headline paragraph to always look different than the rest of the paragraphs in a news article for example.

**General Sibling**

This is like the adjacent sibling except it does not need to immediate follow. The syntax here is y~x. So h1~p would target all paragraphs on the same level as the h1 element, not just the one immediately after it.

## Activity

At this point you have all the tools you need to make a basic web page and change its looks. Draw a simple layout on paper, then try to duplicate it using HTML and CSS.

## Review

Having completed this sub-module you should be able to create rules to select pretty much any element or set of elements on a page. Sometimes you may need to add classes to elements where that makes sense but that is no big deal.