Introduction to HTML & CSS

YOUR INSTRUCTOR TODAY

Pearl Chen
pearl [at] karma-laboratory.com
klab.ca/+ // @pearlchen

(Interactive code slides thanks to CoderDeck)

Course Outline

Morning

- Make sure your computer is setup
- Web development breakdown
- Dive into HTML

Afternoon

- CSS basics
- Getting your website online
- Upload your website

Development Environment

4 tools you need to get started

1. Web Browser

Choose a modern web browser that has good development tools to help make building web pages easier.

Google Chrome

(If you are unable to install Chrome:
install Firefox and Firebug .)

2. Code Editor

Although you could write code in Notepad or TextEdit, choose a text editor with code colour highlighting and other helpful features.

There are many, *many* free and paid options. I recommend this one that's free to evaluate:

Sublime Text

3. Hackasaurus X-Ray Goggles

Use this bookmarklet to "inspect" any webpage and see how it's built. Click and ***drag*** the orange box into your bookmarks bar to install.

(If you can't see your bookmarks bar, use Ctrl/Cmd+Shift+B in Chrome to toggle visibility.)

X-Ray Goggles

(from Hackasaurus by Mozilla)

4. File Transfer Program (FTP)

Do you already have your own hosted website? FTP software will let you copy the files you work on today to somewhere where the rest of the world can see them!

CyberDuck

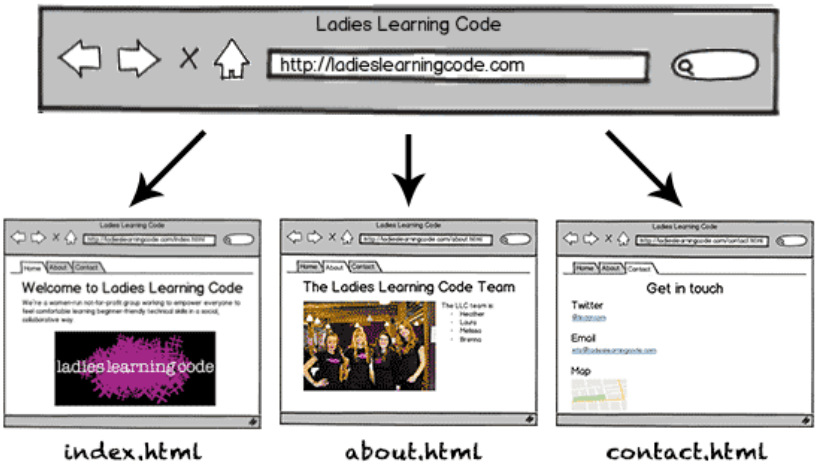
Again, there are many, many free and paid options.
FileZilla is good when you're more advanced.

Organizing your files

Websites, webpages, and file structures

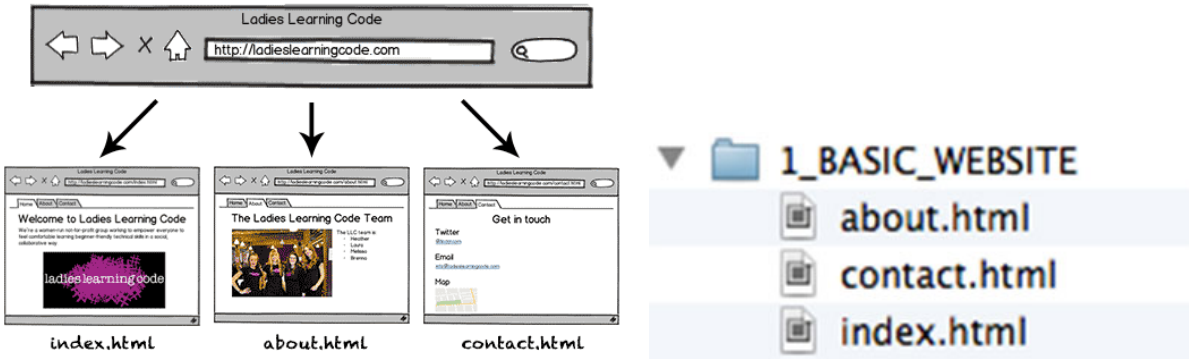
What is a website?

A ***website*** is made up of many ***webpages***.



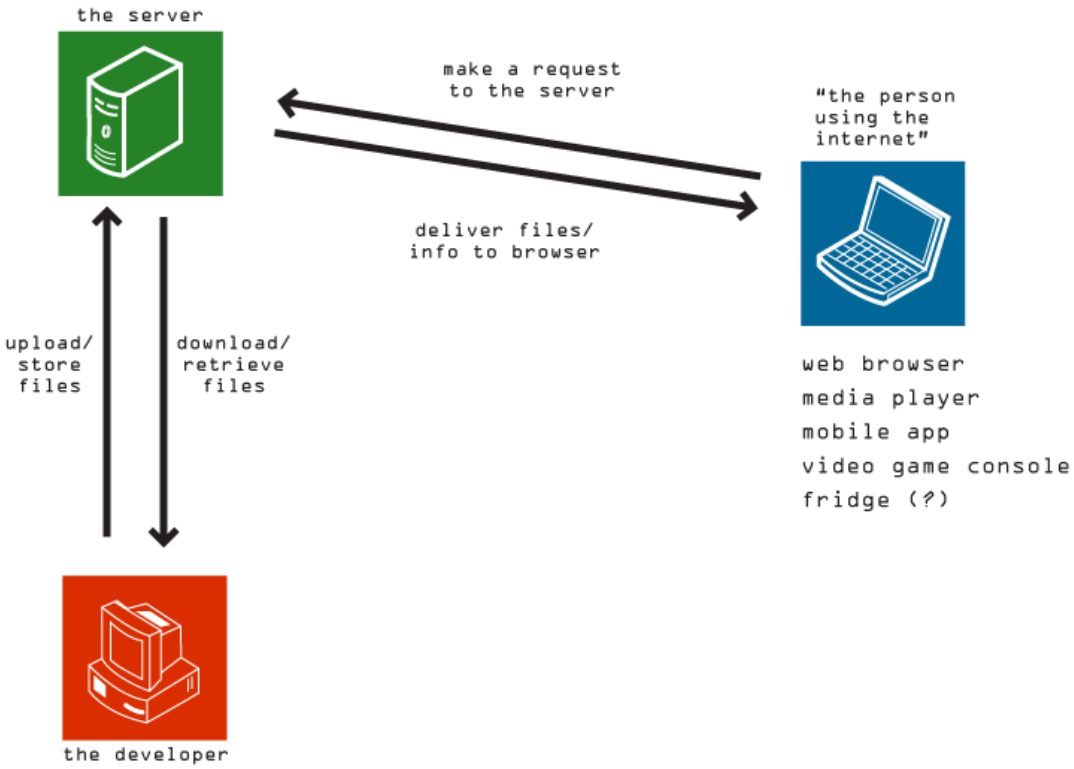
What is a website?

These webpages closely match a file folder structure.



What is a website?

Essentially this folder is located on a dedicated computer (also known as a ***server***) that ***hosts*** websites.



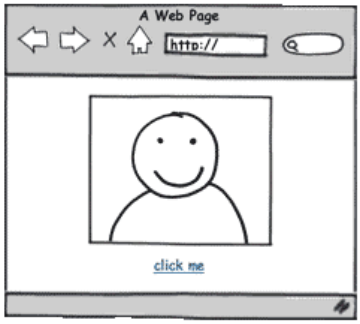
Good file naming habits

- Keep your file names in **all lowercase**.
  `about.html` instead of `About.html` or `ABOUT.html`
- Avoid `s p a c e s` in your file name. Use underscores instead.
  `business_hours.html` instead of `business hours.html`
  (The browswer will interpret it as `business%20hours.html`)
- For the same reason, avoid symbols like `# & *` in your file name.
- Keep your file names concise, yet meaningful.
  `about.html, contact.html` instead of `page1.html, page2.html`

A basic webpage

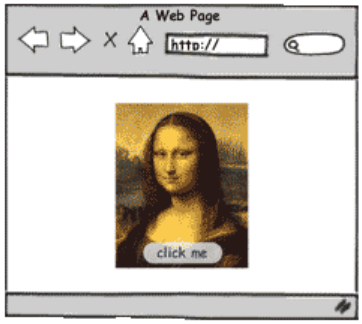
A webpage is mainly built using 3 web languages:

HTML (Hypertext Markup Language)
is the markup language.



It should define the **content**.

CSS (Cascading Style Sheets)
is the style sheet language.



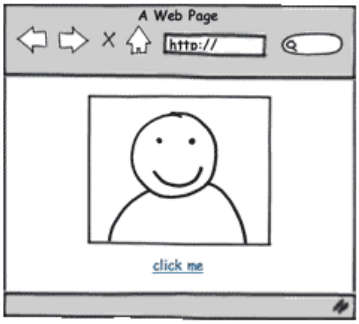
It should define the **presentation**

JavaScript
is the programming language.

It should define **behaviour**.

Dive into HTML

The content layer



What is HTML?

HTML was created for ***web browsers*** to easily read (not humans).

So this makes perfect sense to a web browser:

```
<!DOCTYPE html><html><head><title></title></head><body></body></html>
```

What is HTML?

Whitespace through indentation (tabs) make your more HTML human-readable.

```
<!DOCTYPE html>
<html>
  <head>
    <title> </title>
  </head>
  <body>
  </body>
</html>
```

(Later on, you might also notice how automatic colour coding in your code editor helps a lot too.)

HTML tags

The structure of a webpage is defined by HTML ***tags***. Angled brackets denote tags, like this:

```
<tag>
```

Think about HTML tags like the beams of a house.
It's what holds up a webpage.

HTML tags

Tags typically come in **pairs** -- an opening tag and a closing tag with a forward slash:

```
<tag></tag>
```

Some examples:

```
<!DOCTYPE html>
<html>
  <head>
    <title> </title>
  </head>
  <body>
  </body>
</html>
```

```
<!DOCTYPE html>
<html>
  <head>
    <title> </title>
  </head>
  <body>
  </body>
</html>
```

HTML tags

Tags typically come in **pairs** -- an opening tag and a closing tag with a forward slash:

<tag></tag>

Practice writing a `body` tag with me here:

HTML tags

Tags can be "nested" within other tags, like this:

<tag><tag></tag></tag>

(This relationship is often referred to using the terms "parent" node and "child" nodes.)

For example:

- `<title>` is nested between opening and closing `<head>` tags so `<head>` is the parent node and `<title>` is the child node.

- Going further, <head> is also (at the same time) a child node of <html>.
- While <html> is the always the "root" node and has **no** parent nodes.
- Notice too that <head> and <body> are on the same level. They are both child nodes of <html>.

```
<!DOCTYPE html>
<html>
  <head>
    <title> </title>
  </head>
  <body>
  </body>
</html>
```

```
<!DOCTYPE html>
<html>
  <head>
    <title> </title>
  </head>
  <body>
  </body>
</html>
```

HTML tags

Tags can be "nested" within other tags, like this:

<tag><tag></tag></tag>

Practice writing a head and body tag nested inside of a html tag with me here:

Remember: The innermost tag must be closed **before** trying to open up a new tag.
This would be incorrect: <html><head><body></html></head></body>

HTML tags

Tags can be "nested" within other tags, like this:

<tag><tag></tag></tag>

Now where does the title tag get nested? Practice here:

```
<html>
  <head>
```

Remember: The innermost tag must be closed **before** trying to open up a new tag.
This would be incorrect: <html><head><title></html></head></title>

HTML in a web browser

If you ever forget the basic tags needed to build a basic webpage, `exercises/barebones.html` is available to copy and use.

Let's open it up in our code editor!

Most computers are set to open `.html` files in the default web browser so **right-click** (Command+click or 2-finger click on Mac) and select the "**open with...**" option.

HTML in a web browser

But remember: HTML tags are like the beams of a house -- they denote structure, not content.)

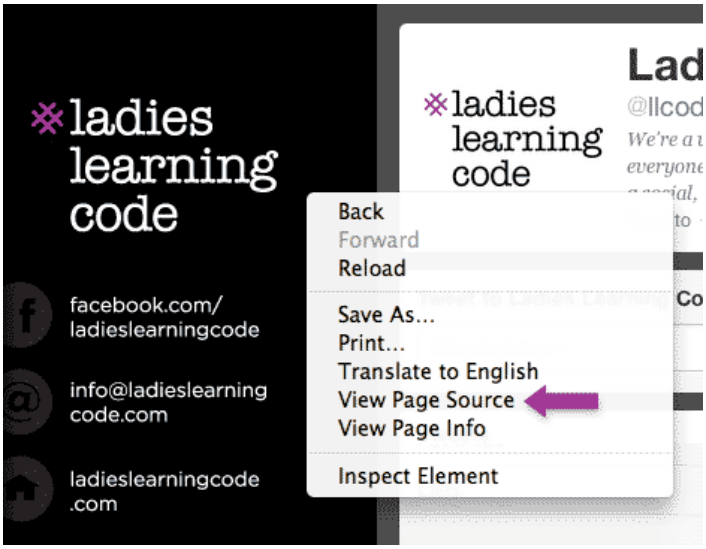
So if I view `barebones.html` in a web browser, I don't see anything interesting yet...

If Chrome is not your default web browser, **right-click** and select the "**open with...**" option.

HTML in a web browser

But I can right-click and select **View Page Source** to see what the HTML is to confirm that it's not a blank file.

(I suggest that you do this yourself any time you see something you like so you can reverse-engineer it.)



Sidenote: HTML Document Types

What about the 1st line?

```
<!DOCTYPE html>
```

This is a ***Document Type*** declaration. `DOCTYPE` indicates to the web

browser what **version** of HTML the file is.

(Consider how old versions of Microsoft Office or Photoshop have problems opening files that were created in later versions of the software.)

For example, it could be an older document type like the ones below, or the newer "HTML5" one that we're using above.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

HTML content

(If HTML tags are the beams of a house, you still need to put up walls and put in furniture for it to be liveable.)

The human-viewable content of a HTML tag goes in between the opening and closing tags like this:

<tag>Content</tag>

```
<!DOCTYPE html>
<html>
  <head>
    <title>Home</title>
  </head>
  <body>
    You are currently looking at index.html.
  </body>
</html>
```

HTML content

The human-viewable content of a HTML tag goes in between the opening and closing tags like this:

<tag>Content</tag>

Practice writing a `body` tag with some human-viewable content here:

<head> vs <body>

Using a copy of `barebones.html`, I'm going to edit it to answer the below questions and prepare you for the following exercise you'll do on your own time.

- What happens when I change the content in the `<title>` tag?
  click here to show answer...
- What's the difference between putting a tag in-between the `<head>` tag versus the `<body>` tag?
  click here to show answer...

Exercise #1 (15 minutes)

1. View/open `exercises/1_BASIC_HTML_WEBSITE/index.html` in Chrome.
2. Make a local copy of `index.html` in the same `1_BASIC_HTML_WEBSITE` folder and rename it `copy.html`.
   (This is so you can always refer back to index.html when needed.)

3. Open `copy.html` in Sublime Text and edit some the content within some of the opening and closing tags.

4. Make sure to save `copy.html` after changes have been made (Ctrl/Cmd+S).

5. View how `copy.html` has changed in Chrome by refreshing the webpage.

6. Explore some of the HTML tags listed on the **next slide** by adding them to your `copy.html`. Re-save and re-view in Chrome.

Exercise #1 - Simple HTML tags

(Don't forget that viewable content goes in-between opening and closing tag pairs. Notice how Sublime Text will hint at you when/where you might not have closed a tag.)

Hierarchy

`<h1>`, `<h2>`, etc to `<h6>` - heading levels 1 to 6
`<p>` - paragraph

Text Formatting

`<b>` or `<strong>` - bold
`<i>` or `<em>` - italics or emphasis
`<center>` - centered

Section Formatting

`<blockquote>` - indented block of code

Input & Forms

`<button>` - clickable button
`<textarea>` - multiline text area for user input

You can practice writing some of these tags in the area below:

Exercise #1 - Exploration Questions

- What are acceptable tags that lend well to being nested and what should never be nested?

- How is whitespace treated? What happens when you use multiple spaces or line breaks?

Got more questions?

Post them to this Google Moderator: http://goo.gl/mod/c4cm

(You'll need to be signed into your Gmail or other Google account.)

Going deeper

Get to know HTML even better

Void (or empty) HTML Tags

Void tags don't format content -- instead they are used to render or embed objects in our page.

These tags **do not** come in pairs like `h1` or `p` tags. You only type the tag once.

In some versions of HTML (specifically XHTML... remember `DOCTYPE`?) they "self-close" with a back slash (/) immediately before the left angled bracket (>). The most often used example of this is the `<br />` tag used to create a **line break**.

```
<p>This text<br />goes over<br />several<br />lines.</p>
```

This text
goes over
several
lines.

Void (or empty) HTML Tags

Void tags

`<hr>` or `<hr />` - horizontal rule
`<br>` or `<br />` - line break
`<input>` or `<input />` - text input boxes

`<img>` or `<img />` - an image **We'll get to this one very soon.*
`<link>` or `<link />` - define a relationship to an external resource like an external style sheets. **We'll get to this one later today.*

Practice some of these tags below:

Attributes

Quick review:

- **HTML tags** define *structure*.
- **Text between opening & closing tags** define *content*.
- **Void HTML tags** render special objects that are sometimes *sort of like content* or *seem structural* depending on the void tag.

Consider the `img` tag. Not very exciting right?

```
<p>Kicking up HTML a notch.</p>
```

Attributes

Sometimes we need to define data that we don't see right away but it's used by the browser to help render special content like hyperlinks, images, or video.

Here's that same `img` tag but now it's using a tag ***attribute*** to tell my webpage where to find a source image:

```
<p>Kicking up HTML a notch.</p>
```

Remember, it's **not** `<img>http://goo.gl/7eSU5</img>`

Working with images

Let's look at that again: images are an example of a ***void tag* with** an ***attribute***.

```
<img src="http://goo.gl/7eSU5" />
```

- `src` is the *attribute*
- `http://goo.gl/7eSU5` is the *value* of that attribute
  Values are typically surrounded by *single or double quotes* since occassionally values may contain spaces.

With an `img` tag, the value of the `src` attribute tells the browser what the *url source* of our image is.

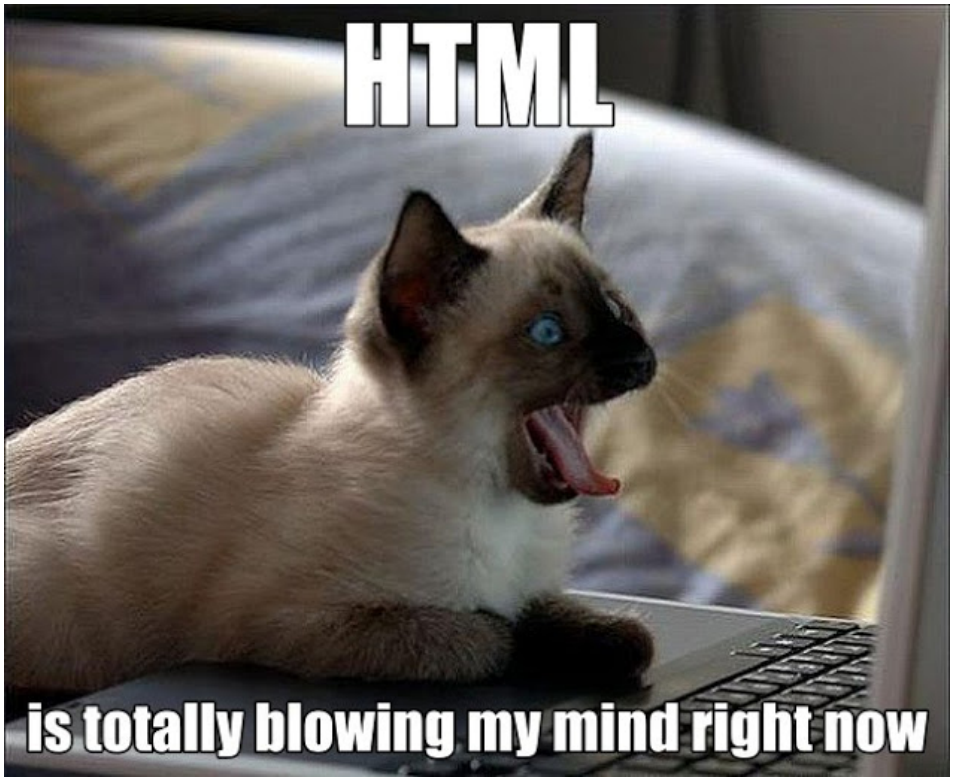
Working with images

Tags can have more than one attribute, some **required** (like `src`) and others are **optional**:

- `width` and `height` tell the browser what dimension to *display* the image at.
  (This is useful so the page doesn't jump around when loading very large images, or while on a very slow internet connection such as dial-up or 2G/3G. Also useful to know that width/height values don't necessarily have to match the image's *actual* width/height.)

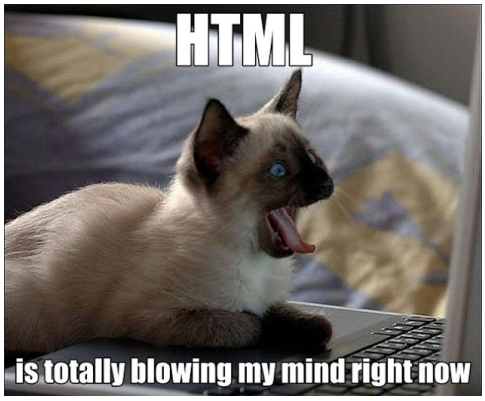
For example, without `width` and `height` attributes, the browser displays image at original size.

```
<img src="http://goo.gl/7eSU5" />
```



Now with a manipulated width and height (about 50% smaller):

```
<img src="http://goo.gl/7eSU5" width="320" height="262" />
```



Working with images

Another optional image attribute:

- `alt` is *alternative text* to display to visually impaired web visitors using screen readers. It also shows up when there's a "broken image" due to typos or moved files.

  (Technically, it's required to be "valid" HTML markup which is why you might see `alt=""` often used.)

```
<img
  src="http://goo.gl/7eSU5"
```

Working with images

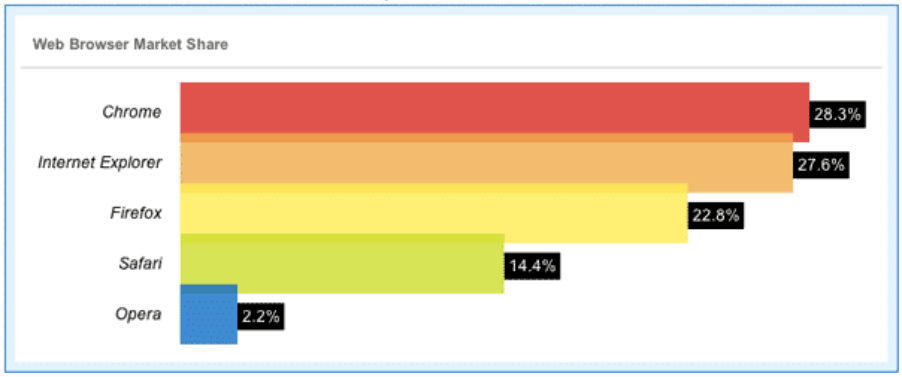
In the `exercises/2_WORKING_WITH_IMAGES` folder, there's an `index.html` file and 3 images of various types.

> To make your web development life easier, turn on the option to see the file extension type in Explorer (Windows instructions) or Finder (Mac instructions).

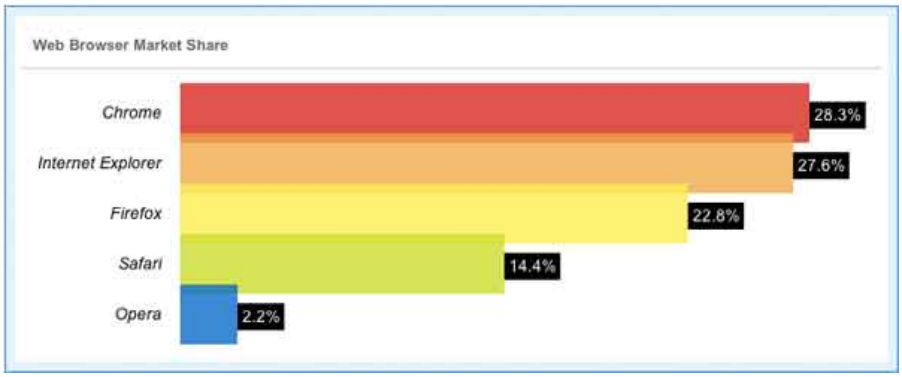
File type advantages for your reference:

- ***gif*** - smaller file size when used for line art, charts, and images with minimal colour variation
- ***jpg*** - smaller file size when used for photographs with lots of rich colours or gradients
- ***png*** - larger file size but has best quality and is great for images with transparent backgrounds

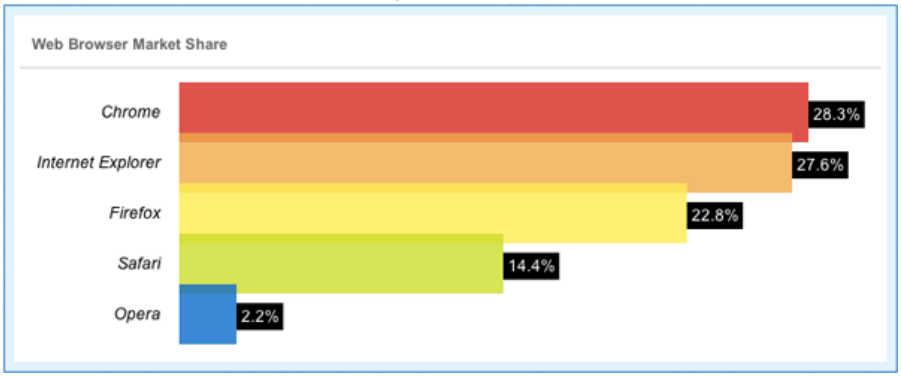
As a gif: 600 x 247, 8kb



As a jpeg: 600 x 247, 9kb



As a png: 600 x 247, 18kb



Working with images (Group exercise - Part 1)

Code along with me:

1. Open `exercises/2_WORKING_WITH_IMAGES/index.html` in Sublime Text. (It's simply a copy of `barebones.html`.)
2. Click on the above link to also view `index.html` in Chrome.
3. Use the `img` tag to embed an image of our fave LOL cat using `http://goo.gl/mod/c4cm` for the value of the `src` attribute.
4. Save and refresh the webpage in the browser to make sure you got it right.

This is an example of an ***absolute path***.

It's "fully resolved" and starts with `http://` much like typing in a url into the browser's URL bar.

Working with images (Group exercise - Part 2)

Code along with me:

1. Return to our `img` tag in `index.html` and let's instead embed `llc.gif` which we have locally.

This is an example of a ***relative path***.

It's based on using the html file that you're working on as the starting point to find your image source.

Right now, `llc.gif` in the **same folder** as `index.html`.

2. Open the `images.html` file in the `img` folder in Sublime Text.
3. Click on the above link to also view `images.html` in Chrome.
4. Note that there's already an `img` tag in `images.html` trying to embed `llc.gif`.

Working with images (Group exercise - Part 3)

To write a ***relative path*** that goes ***up*** a folder, start path with "`../`".

```
<img src="../llc.gif" />
```

Code along with me:

1. Return to our `img` tag in `images.html` and add `../` infront of `llc.gif`.

An aside: `../` can be used multiple times if you have many nested folders.

```
<img src="../../../../llc.gif" />
```

How many folders does this relative path go up?
click here to show answer...

Working with images (Group exercise - Part 4)

However, it's best practice to organize your images into its own folder to keep your website's file structure tidy.

We tend to also organize JavaScript files into their own "js" folder and CSS files into their own "css" folder.

So let's **move all of our images into the `images` folder** of `2_WORKING_WITH_IMAGES`.

Uh oh! Now our image will be broken in `index.html` since the image is no longer in the same folder. Let's fix that...

Working with images (Group exercise - Part 5)

To write a ***relative path*** that goes ***into*** a folder, start path with the **folder name** followed by a backslash (/).

```
<img src="images/llc.gif" />
```

1. Return to `index.html` and add `images/` infront of `llc.gif`.

An aside: You can go into multiple nested folders:

```
<img src="folder1/folder2/folder3/llc.gif" />
```

Hyperlinks

The <a> *anchor* tag is used to tell the browser that we want to link to somewhere else on our site or on the web.

The a tag comes in pairs and the content between the opening and closing tags is what is displayed as the link text (typically underlined).

```
Visit <a>Ladies Learning Code</a>.
```

Visit Ladies Learning Code.

But wait! This link doesn't take me anywhere!

Hyperlinks

We need a way of setting the URL (e.g. http://ladieslearningcode.com) that the browser will request when someone clicks on the link.

Once again, we need to use ***attributes***.

```
Visit <a href="http://ladieslearningcode.com">Ladies Learning Code</a>.
```

Visit Ladies Learning Code.

- `href` is the *attribute*
- `http://ladieslearningcode.com` is the *value* of that attribute
- And remember that the **clickable text goes in between** the opening and closing a tags.

Exercise #2 (10-15 minutes)

1. Continue using `index.html` from the `2_WORKING_WITH_IMAGES` folder in Sublime Text.

2. Create a text link using the <a> tag that points to "http://ladieslearningcode.com".

3. Using your knowledge of the <a> tag, try to create a *clickable image* using both the <img> and <a> tags. Clicking on the image should open up "http://ladieslearningcode.com".
   (Hint: This involves nested HTML tags.)

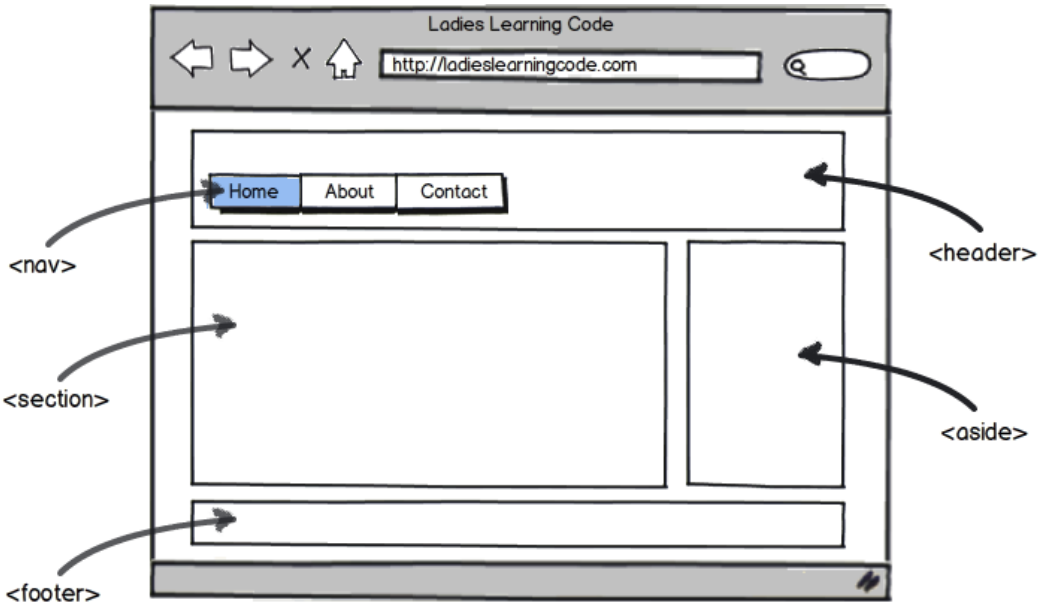
Make sure to save `index.html` after changes have been made (Ctrl/Cmd+S) and before confirming your edits in Chrome.

Got questions?

Post them to Google Moderator: http://goo.gl/mod/c4cm

Getting ready for Exercise #3

We're eventually aiming to have a **3-page** website that uses a standard 2-column webpage layout.



Getting ready for Exercise #3

Code along and review with me!

Make a new `index.html` in the `exercises/website` folder.

1. What's the **very first line** we need to type?
2. What **three** HTML tags should every website have?
3. What HTML tag do I need to update the browser tab with a descriptive tab name? And where does it go?

Here is the start of your top navigation menu.

(Don't worry, `<nav>` is nothing too special but we'll use it later in CSS.)

```
<nav>
  Home About Contact
</nav>
```

4. In between which tag do I paste this code since it's content that should be **visible to the user**?

Exercise #3 (now until lunch)

On your own:

1. Eventually there will be `about.html` and `contact.html`. Use the `<a>` tag with a `href` attribute to link all your pages together.
2. Based on `index.html`, create those 2 new files for About and Contact in the same `website` folder as `index.html`.
3. Make sure that all your webpages have descriptive and unique `<title>`'s.
4. Confirm that you can click between all the pages using the navigation links. Fix them if needed.
5. Update `about.html` with a picture and bio.
   (You can even try adding a YouTube video based on the notes on the next slide.)
6. Update `contact.html` with contact info.
   (You can even try adding a Google Map.)
7. **Don't** update `index.html` any more!
   We'll get to that **after** lunch.

Got questions?

Post them to Google Moderator: http://goo.gl/mod/c4cm

Bonus: Add video to your webpage

As time goes on, it has become easier and easier to embed multimedia content into websites.

From YouTube, you can find an **Embed Code** to put into your website (click on "Share" button below video, then "Embed") and YouTube's servers do all the rest of the work via the `iframe` tag.

```
<iframe width="560" height="315" src="http://www.youtube.com/embed/tlnFyjpV5Wo"
frameborder="0" allowfullscreen></iframe>
```

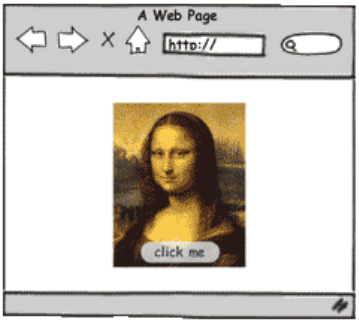
Adding a Google Map is similiar. Click here for instructional screenshot.

HTML reference

- Download a cheatsheet of HTML tags. (Copy in the `cheatsheets` folder.)
- Try looking up individual tags using the HTML sections of Mozilla's Developer Network or the W3C for further breakdown.
- For further exploration, use the Hackasaurus X-Ray Goggles to look at any webpage.
- And don't forget the "view source" option of your web browser!

CSS basics

The presentation layer



Why use CSS?

- Seperates content (text) from presentation (text colour).
- The "cascading" part of CSS means that it (should) help avoid redundancy in styling your web documents.

Inline CSS using the `style` attribute

The `style` ***attribute*** can be added to any visible HTML tag.

```
<h1>Welcome!</h1>
<p>This is my website.
```

Using selectors to apply CSS

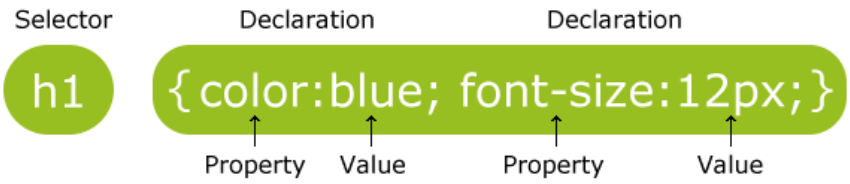
Here's the same result but using CSS ***selectors***.

The `<style>` tag (placed between the `<head>` tags) is where CSS commonly goes.

```
<h1>Welcome!</h1>
<p>This is my website.
```

CSS selectors and properties

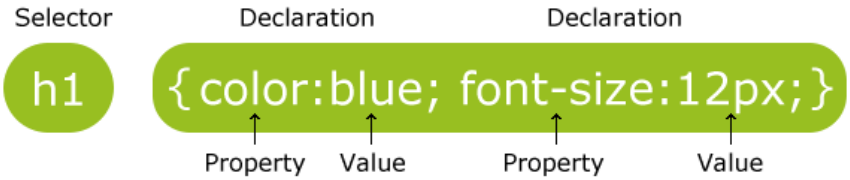
Here's a CSS declaration dissected:



- A **basic *selector*** will match its HTML tag name. In this case, all content that is a `<h1>` will be applied this style.
  (We'll get into more complex ones later.)
- A ***property*** is a special label for some visual aspect of the selector such as color or size. You can see a list of them in this cheatsheet.
- The property **value** is specific to the property type. For example, `color` properties accept colour values like `blue` or `#FFFF00`. `font-size` properties accept units of measurements like `12px` (where "px" stands for pixels).

CSS selectors and properties

Here's a CSS declaration dissected:



- Use **curly braces** (`{}`) to hold a list of the property:value sets.
- You can define multiple property:value sets but you must end each one with a **semi-colon** (`;`).

Changing the page's background

colour

View exercises/3_CSS/index.html in your browser. Notice how we're using a transparent PNG so it's hard to read all the white letters on a white background.

Let's add a <style> tag between the <head>. Then add this:

```
body { background-color: gray; }
```

Changing the font colour

Now the text is a bit hard to read on the gray background. Because the <body> tag is the highest possible visible HTML tag, the body selector will also allow us to target the font color on the *entire* page.

```
body { background-color: gray; color: white; }
```

Changing the font family

The web has a number of fonts that are considered "web safe". Some of the most used fonts are:

- Arial
- Verdana
- Times New Roman
- Georgia
- Courier

Let's add a bit of contrast to our headers by using the Courier font on them.

```
h1 { font-family: Courier; }
h2 { font-family: Courier; }
```

Changing the font family

Not all fonts are installed on all computers so it's recommended that you use a list of fonts. If the first font in the list isn't available, it will try to use the next.

```
body { background-color: gray; color: white; font-family: Helvetica, Verdana, Arial, sans-serif; }
```

(Notice how the font-family for the headers are still Courier? That's the "cascading" part of Cascading Style Sheets in action. Inline styles take precedence, then innermost tags, then outer tags.)

Changing the font size

Font sizes are typically defined in pixels (px) or "ems" (em).

- 20px
- 45px
- 0.5em
- 1em

Headers have default font-sizes defined by the browser. Let's override them.

```
h1 { font-family: Courier; font-size: 25px; }
h2 { font-family: Courier; font-size: 22px; }
```

(When you've got more time, read this article for more info on font unit sizes.

Changing the font weight

In its simplest form, there are 2 weights:

- normal
- **bold**

By default, headers are already bolded. Let's remove the bold by applying the *normal* font weight on them.

```
h1 { font-family: Courier; font-size: 25px; font-weight: normal; }
h2 { font-family: Courier; font-size: 22px; font-weight: normal; }
```

Formatting CSS

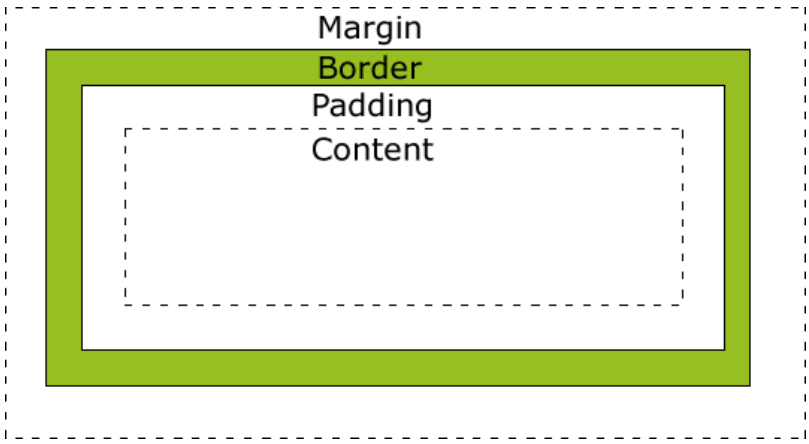
As with HTML, extra whitespace and line breaks add to readability.

Our current CSS can be written like this:

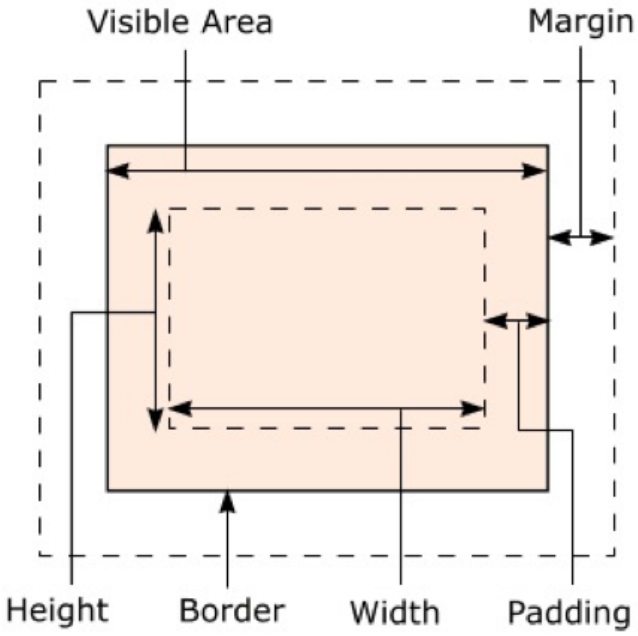
```
<style>

  body {
    background-color: gray;
    color: white;
    font-family: Helvetica, Verdana, Arial, sans-serif;
  }

  h1 {
    font-family: Courier;
    font-size: 25px;
    font-weight: normal;
  }

  h2 {
    font-family: Courier;
    font-size: 22px;
    font-weight: normal;
  }

</style>
```

The "Box Model"

Most HTML elements on a webpage look like this to the browser:



This is the same but visualized differently:



Example: Let's make our links look more button-like

To add a border you need to set 3 properties: `border-weight`, `border-style`, and `border-color`.

You can define them seperately, or using shorthand.

Note: CSS comments use the format `/* comment */`

```
<style>
a {
```

Example: Let's make our links look more button-like

To add padding you set 4 properties: `padding-top`, `padding-left`, `padding-bottom`, and `padding-left`.

You can define them seperately, or using shorthand (in a clockwise direction).

```
<style>
a {
```

Example: Let's make our links look more button-like

Add a background colour and change the font colour.

To get rid of the underline, use the `text-decoration` property.

```
<style>
a {
```

Example: Let's make our links look more button-like

To add margins you set 4 properties:
`margin-top`, `margin-left`, `margin-bottom`, and `margin-left`.

You can define them seperately, or using shorthand (in a clockwise direction).

```
<style>
a {
```

Example: Let's make our links look more button-like

Here's the final button code!

```
<style>
a {
```

Advanced topic: To get it looking even more like a button, use the `border-radius` property.
(We won't get into browser pre-fixes today but do know that newer CSS features require you to re-state the property for each browser vendor.)

CSS interactions & pseudo selectors

CSS is also able to handle some basic UI interactions in our sites, such as allowing a mouse over or "hover" state on various elements. This is frequently done with items such as links.

As part of your selector, you need to use the `:hover` ***pseudo selector***.

```
<style>
a {
```

CSS classes

Wait! What happens when we add another link to the page and it's not part of our navigation? Putting all the styles inline wouldn't be wise...

```
<style>
a {
```

Now that's a lot of CSS! What if you change you mind and decide that the background color should be blue now? Plus you can't apply pseudo selectors like `:hover` inline.

CSS classes

If you know that several elements are going to have similiar styles, use CSS classes! Once you define one CSS class, you can apply it over and over again!

- Use a **period** (`.`) to define CSS selectors that are classes.
  e.g. `.nav_button`
- Then apply that class to any element using the `class` attribute.
  e.g. `class="nav_button"`

```
<a class="nav_button">Home</a>

<style>
.nav_button {
  font-family: Georgia;
}
</style>
```

CSS classes

Let's apply a "nav_button" CSS class to our current navigation links:

```
<style>
a{
```

External CSS

Instead of duplicating your `<style>` content for every webpage, make one single CSS document ending with a `.css` extension and link to it in the `<head>` tag instead of (or in addition to) a `<style>` tag.

```
<head>
  <link rel="stylesheet" href="styles.css" type="text/css" />
  <style>
  body {
    font-size: 14px;
  }
  </style>
</head>
```

Inside styles.css, there is **no** `<style>` tag:

```
body {
  background-color: black;
  color: white;
  font-family: Helvetica, Arial, sans-serif;
}
h1 {
  font-family: Georgia, "Times New Roman", serif;
}
```

Exercise #4 (20 minutes)

Use this time to apply CSS to your 3-page website.

- Update your navigation in `index.html` to look more button-like.
- Create a single `styles.css` file and link to this external stylesheet from all 3 webpages using the `<link>` tag. Consolidate all of your CSS into this external stylesheet.
- If you've got extra time, try out some new CSS properties from the cheatsheet and other resources listed on the CSS resources slide.

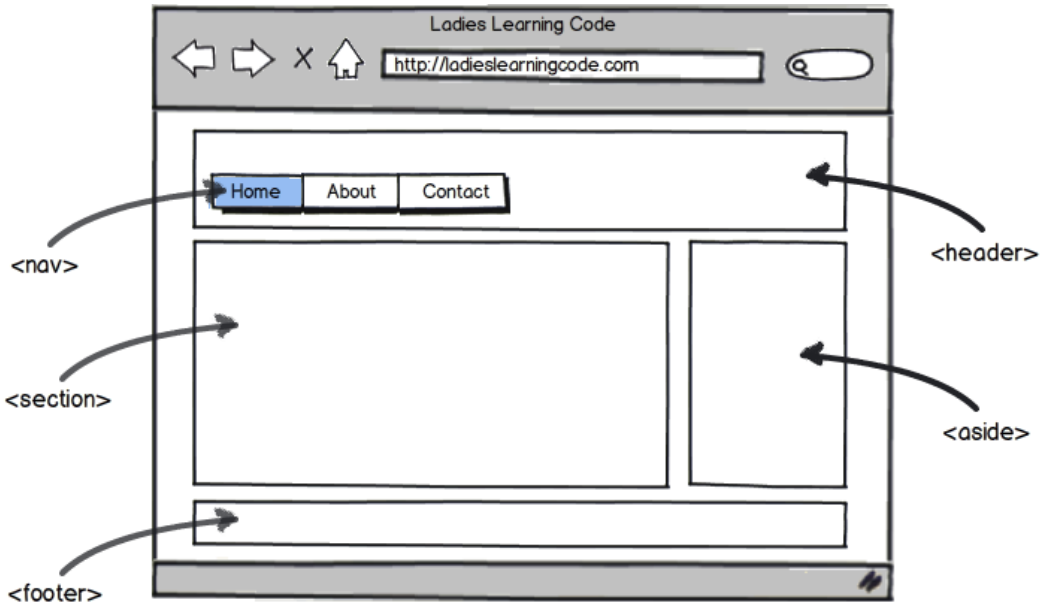
Got questions?

Post them to Google Moderator: http://goo.gl/mod/c4cm

Laying out webpages with CSS

Layout is one of the most challenging things to do in CSS because it requires you to think about the whole picture and plan ahead.

To creating a standard 2-column layout requires us to give dimensions to some of our content, and position it accordingly.



Generic Container Tags

Sometimes content you might want to group together (and treat differently with CSS) shouldn't be defined as a `h1` or `h2` or even a `p` paragraph. There are two kinds of generic containers available:

- `span` - An ***inline*** container.
  Good for highlighting keywords like I did in this bullet.
- `div` - An ***block*** container.
  Much like a paragraph, it creates a line break above and below to create a box.

```
<div>I'm a div!</div>
<div>I'm another div!</div>
```

I'm a div!
I'm another div!

Semantic HTML Tags

Historically, when making non-trivial websites, it required nesting lots of `div` tags within `div` tags (within `div` tags!) like this:

```
<div>
  <div>
    <div>Home</div>
    <div>About</div>
    <div>Contact</div>
  </div>
</div>
<div>
  Welcome to my blog!
  <div>[...Blog Post...]</div>
  <div>[...Blog Post...]</div>
</div>
<div>
  [...Sidebar Ad...]
</div>
<div>
  Copyright 2012
</div>
```

Semantic HTML Tags

With HTML5, there are an increasing number of semantic tags that better describe larger blocks of content such as: `header`, `footer`, `nav`, `section`, `article`, and `aside`.

```
<header>
  <nav>
    <div>Home</div>
    <div">About</div>
    <div">Contact</div>
  </nav>
</header>
<section>
  Welcome to my blog!
  <article>[...Blog Post...]</article>
  <article>[...Blog Post...]</article>
</section>
<aside>
  [...Sidebar Ad...]
</aside>
<footer>
  Copyright 2012
</footer>
```

Semantic HTML Tags

This helps ***web crawlers*** from search engines like Google or Bing better understand your content.

Even better, it helps **you** from missing a closing `</div>` tag and makes your HTML easier to read for everyone.

Preparing for our group exercise

Using `index.html` from the `website` folder, let's update it with the code below.

It should replace everything in between the opening and closing <body> tags.

```
<div>
  <header>
    <nav>
      <a href="index.html">Home</a>
      <a href="about.html">About</a>
      <a href="contact.html">Contact</a>
    </nav>
  </header>
  <section>
    Welcome to my blog!
    <article>[...Blog Post...]</article>
    <article>[...Blog Post...]</article>
  </section>
  <aside>
    [...Sidebar Ad...]
  </aside>
  <footer>
    Copyright 2012
  </footer>
</div>
```

Assigning widths

It is important to know the overall width of your site so you can select column widths for the main <section> and the <aside> sidebar that will make sense.

Let's go ahead and set the <header> and <footer> to be 960 pixels wide. We'll add a gray border too, so we can see what's going on.

```
header {
  width: 960px;
  border: 1px solid gray;
}

footer {
  width: 960px;
  border: 1px solid gray;
}
```

Assigning widths

If our site is 960px wide, we need to ensure that our columns don't exceed 960 - **including margins!** (e.g. 640 + 20 + 300 = 960)

```
section {
  width: 640px;
  padding: 30px;
  border: 1px solid gray;
}

aside {
  margin-left: 20px;
  width: 300px;
  border: 1px solid gray;
}
```

PRO TIP: Historically, padding wrecked havoc on our math. To avoid headaches, use the box-sizing property for every website you make to make sure your math adds up.

```
/* apply a natural box layout model to all elements */
* {
  -moz-box-sizing: border-box;
  -webkit-box-sizing: border-box;
  box-sizing: border-box;
}
```

Getting columns to be side-by-side

Block-style HTML containers like `<p>`, `<div>`, `<section>`, and `<aside>` automatically start on a new line.

To get our two columns to be side-by-side, use the `float` property. Values can be `left`, `right`, or `none`.

```
section {
  width: 560px;
  padding: 30px;
  border: 1px solid gray;
  float: left;
}

aside {
  margin-left: 20px;
  width: 300px;
  border: 1px solid gray;
  float: left;
}
```

Clearing floats

You may notice that floats have the ability to affect the rest of the webpage in unexpected ways. It's important to always clear your floats.

One way is to use the `clear` property on an element that comes *directly* after the containers that are floating. Values can be `left`, `right`, or `both`.

```
footer {
  width: 960px;
  border: 1px solid gray;
  clear: both;
}
```

If the elements that are floating are enclosed in another container, you can you apply `overflow:auto` to the parent container:

```
nav {
  overflow: auto;
}
```

Another way (when all else fails) is via a line break tag with a clear attribute: `<br clear="all" />`

CSS id's

CSS classes are great when you want to re-apply the same style to multiple HTML elements but sometimes you need to only target an HTML container **once**. This is where an ***ID*** is useful.

- Use a **hash** (#) to define CSS selectors that are IDs.
  e.g. `#centered_on_page`
- Use the `id` attribute to apply the styles to that element.
  e.g. `id="centered_on_page"`

Important! You should only have **one** instance of an ID per page. Any more can cause unexpected things to happen with your CSS -- and later, JavaScript.

```
<style>
  #centered_on_page {
```

CSS id's

Let's add `#centered_on_page` to `styles.css` and apply the id to the `div` container immediately after the opening `<body>` tag in `index.html`.

```
#centered_on_page {
  margin: 0 auto 0 auto;
  width: 960px;
}
```

```
<body>
  <div id="centered_on_page">

  </div>
</body>
```

Custom colours

Often, the CSS colours "gray", "blue", or "purple" are boring and limiting. You can define your own custom colour using hex colours.

```
h1 {
  color: #026873; /* a unique shade of blue */
}
```

Image source: http://websiteincite.com/create-your-website/hexadecimal-color-codes

Chart of many hex colours: http://www.visibone.com/color/chart_847.gif

Hands-on time (now until end)

Use this time to apply CSS to your 3-page website. Review the afternoon's slides and experiment with some of the properties on the cheatsheet. If you're stuck, here are some suggestions:

- Apply `widths` and `floats` to `<section>` and `<aside>` to make a standard 2-column layout.
- Update your `about.html` and `contact.html` with some of your own CSS.
- Apply some custom colours to your text in any of your web pages.
- Research the `background-image` property and add a background image to webpages.

Got questions?

Post them to Google Moderator: http://goo.gl/mod/c4cm

Bonus: Try out web fonts

When you get bored of Arial and Georgia, go to the Google Web Fonts website and find a font that you like.

1. Click on the "Quick-use" button for that font.
2. Scroll down to find the supplied `<link>` tag. Copy and paste it into your own HTML file.
3. Integrate the fonts into your CSS by using their supplied `font-family`. For example, add their `font-family` to your `h1` CSS selector.
4. Make sure your files are saved and view your web page in the browser. New fonts are that easy!

Bonus: CSS patterns for fun

CSS is great because it can reduce the number of image files you need to create. You can even create patterns purely out of CSS!

1. Click on a pattern that you like.
2. Copy the supplied CSS.
3. Instead of an "ad" in your sidebar, paste the CSS into your `aside` selector. Or apply it to your `body` selector to change up the entire page.
4. Important: Due to the differences in CSS support in web browsers, you will need to duplicate the `background-image` property-value set and replace `linear-gradient` with `-webkit-linear-gradient`.

```
/* Based on Cicada stripes from http://lea.verou.me/css3patterns/#cicada-stripes */
background-color: #026873;
background-image: linear-gradient(90deg, rgba(255,255,255,.07) 50%, transparent 50%),
                  linear-gradient(90deg, rgba(255,255,255,.13) 50%, transparent 50%),
                  linear-gradient(90deg, transparent 50%, rgba(255,255,255,.17) 50%),
                  linear-gradient(90deg, transparent 50%, rgba(255,255,255,.19) 50%);
background-image: -webkit-linear-gradient(0deg, rgba(255,255,255,.07) 50%, transparent 50%),
                  -webkit-linear-gradient(0deg, rgba(255,255,255,.13) 50%, transparent 50%),
                  -webkit-linear-gradient(0deg, transparent 50%, rgba(255,255,255,.17) 50%),
                  -webkit-linear-gradient(0deg, transparent 50%, rgba(255,255,255,.19) 50%);
background-size: 13px, 29px, 37px, 53px;
```

CSS reference

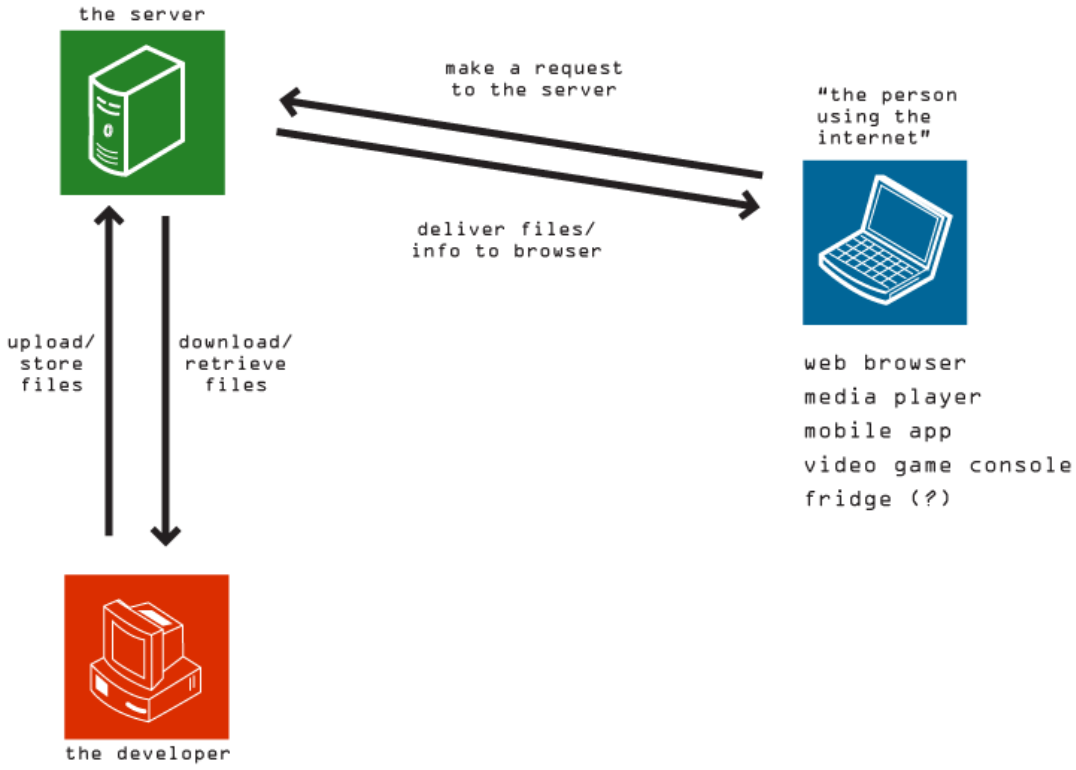
- Download a cheatsheet of CSS selectors and properties.
- Try looking up individual selectors and propertie using the CSS section of Mozilla's Developer Network for further breakdown.
- Read articles from A List Apart.
- And don't forget the "inspect element" option!

Getting your website online

Using FTP

Déjà vu: What is a website?

Remember how a website is essentially a folder located on a dedicated computer (a ***server***)?



Using FTP to put your site online

If you have access to web hosting and would like to put your site online, there are a few vital pieces of information you need:

1. FTP Address

2. Username
3. Password
4. Any special security info
5. Which directory to transfer your files to

Where to get this information

This information will generaly be provided to you by your hosting service provider.

There ARE conventions regarding what directory your files go into once you have connected to your server via FTP:

- www
- public
- public_html
- htdocs

Transferring your files

Once you have identified where your files need to go, it is often just a matter of dragging and dropping into your FTP program.

Once this is done, you can check your web address to see how things are looking!

Additional Resources

Some tips and tricks for the future

Books

HTML and CSS: Design and Build Websites by Jon Duckett

Validating your HTML

One of the best ways to make sure that your HTML is being done correctly is to run it through an HTML validator on occasion.

You can find the HTML validator at http://validator.w3.org/

One last peice of advice:

Clear your cache

Hitting refresh and still seeing old stuff? The browser cache is the rip-your-hair-out *gotcha!* of every web developer.

Instructions for every browser (including the archaic one your client is

running): http://www.wikihow.com/Clear-Your-Browser's-Cache

Also try out "Incognito Mode" in Chrome: Ctrl/Cmd+Shift+N

Thank you!