Intro to HTML

Prerequisites

- Ruby Web Servers

What you’ll learn

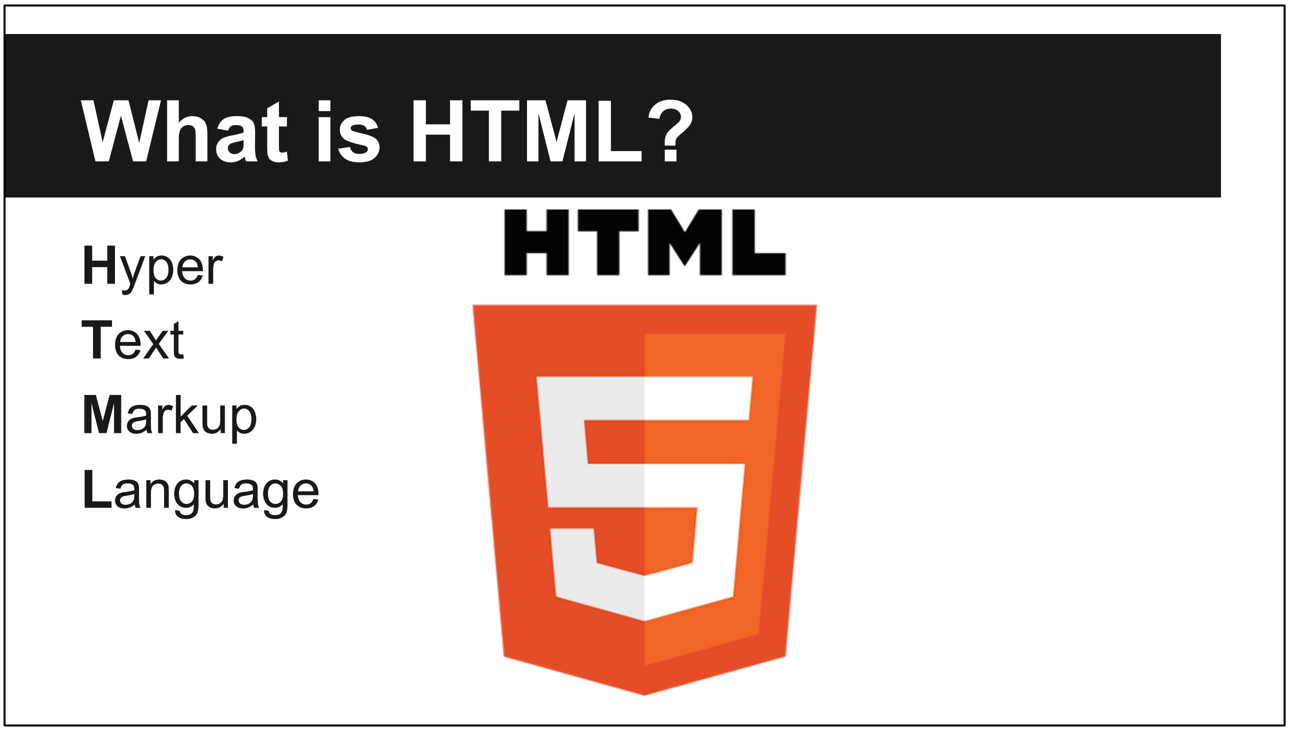
- History of HTML
- W3C & the Specification process
- HTML Validation
- HTML Syntax
- Some HTML tags
- HTML Nesting
- Debugging HTML
- Semantic HTML (if there’s time)

*

What you won’t learn

Coming Soon:

- CSS
- Javascript



What is HTML?

Hyper
Text
Markup
Language

HTML

HTML stands for **H**yper**T**ext **M**arkup **L**anguage.

A markup language is *not* a programming language. That’s a common mistake.

Computers didn’t exist until the 1940s. The first programmer, a *woman* named Ada Lovelace, worked in the 1840s (a century earlier).

But people have been “marking up” documents for centuries. The Gutenberg Bible, the first book ever mass produced for the public, was “marked up” in the 1450s. The Chinese were “marking up” text 1000 years ago.

Publishing

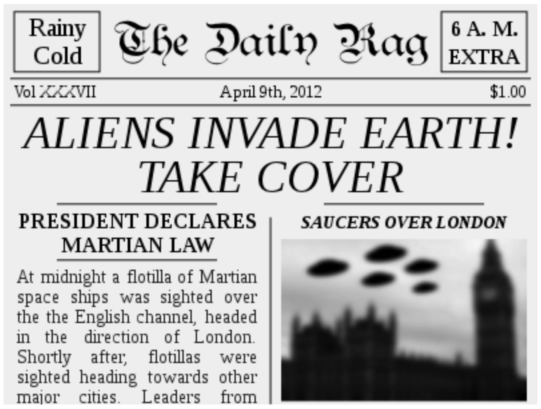
Aliens Invade Earth!

Take Cover

Justin Bieber!

Markup languages come from the publishing industry. They’re how you take plain text, like this.

Publishing



Rainy Cold | The Daily Rag | 6 A. M. EXTRA

Vol XXXVII April 9th, 2012 $1.00

ALIENS INVADE EARTH! TAKE COVER

PRESIDENT DECLARES MARTIAN LAW

At midnight a flotilla of Martian space ships was sighted over the the English channel, headed in the direction of London. Shortly after, flotillas were sighted heading towards other major cities. Leaders from

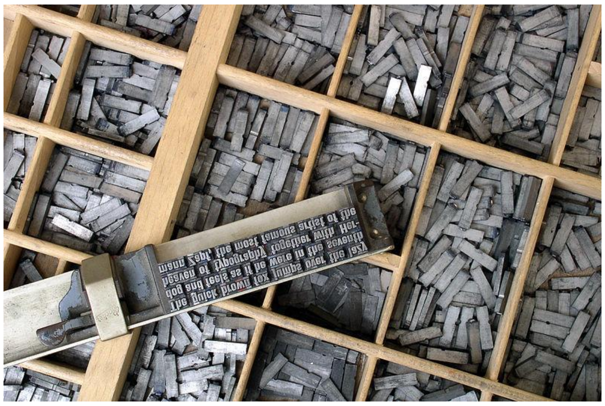
SAUCERS OVER LONDON

VANITY FAIR
WIKIGATE
THE TWISTED INSIDE STORY of HOW JULIAN ASSANGE SPILLED the GOVERNMENT'S BIGGEST SECRETS
The Kennedy INAUGURATION:
J.F.K., SINATRA, and the 1961 BASH THAT LAUNCHED CAMELOT
Huffington Poach?
THE ALLEGED RIP-OFF BEHIND a $350 MILLION ONLINE PHENOMENON
SURFING The World's Giant Waves
Warren Buffett's RISKIEST BET YET
Justin BIEBER!
Is This the Adorable, Inescapable Face of 2011?

… and turn it into this.

To transform plain text into a newspaper or magazine cover, someone has to take each letter and organize it into a layout.

Typesetting



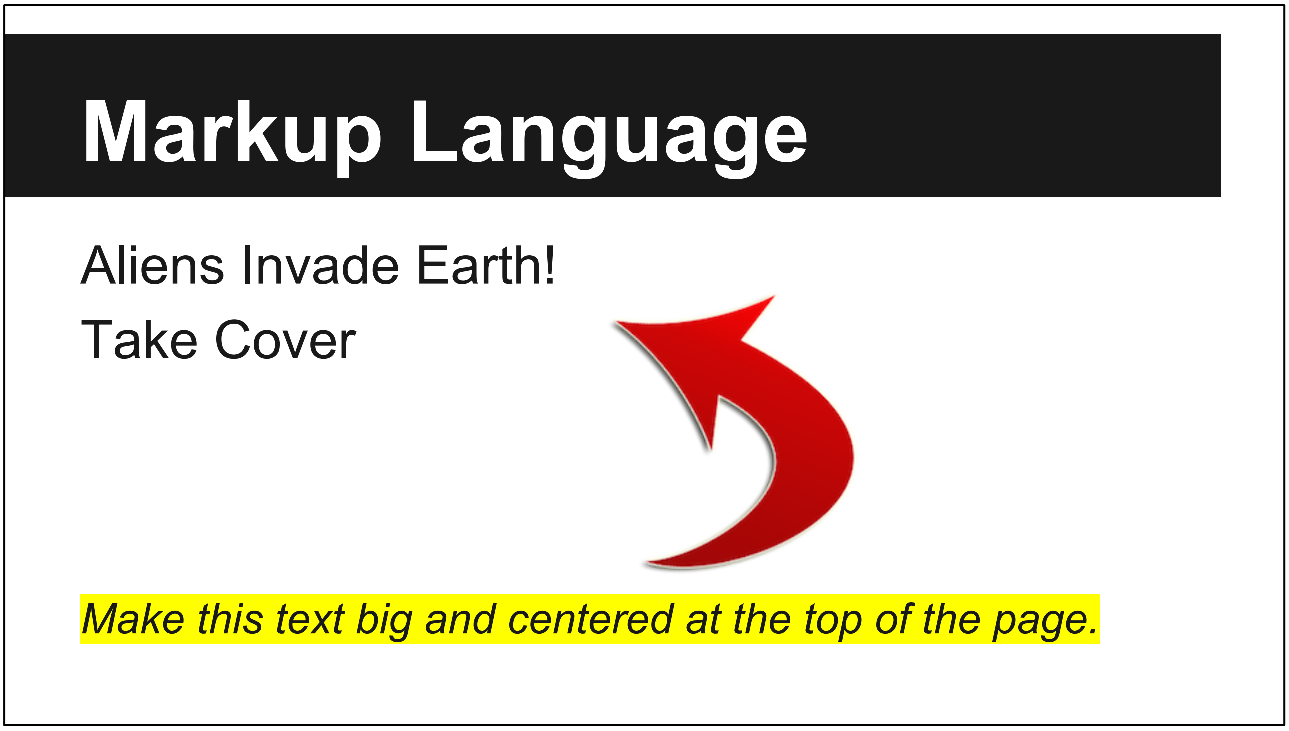
This is what those letters look like. Using a process called “movable type” a person called a “typesetter” would lay out metal “types”, one for each character, onto a “forme”.

Forme

A “forme” like this. A “forme” is a big piece of wood or metal. A typesetter lays types out onto a forme to define a layout.

When the “forme” is complete, it’s dipped in ink and used to make an impression on a piece of paper.

This process of using “movable type” - movable components to reproduce letters and punctuation - predates computers by centuries. The European mechanical printing press - the one that produced the Gutenberg Bible in the 15th century - is regarded as one of the key factors fostering the Renaissance. In fact, this process is over 1000 years old, dating back to China in the 10th and 11th centuries.



Markup Language

Aliens Invade Earth!

Take Cover

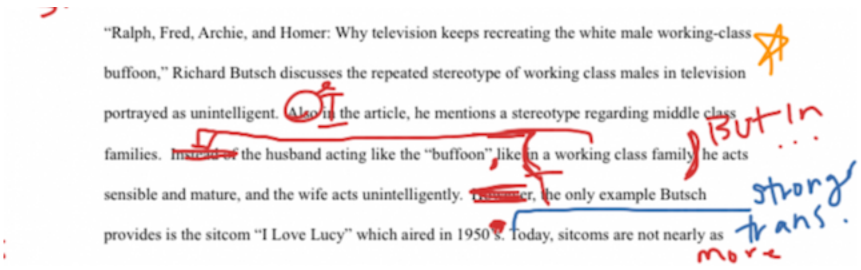
Make this text big and centered at the top of the page.

Someone has to write little notes letting the typesetter know which types (which letter blocks) to use and how to lay them out. This is called “marking up” a manuscript. And the people who wrote these little notes were called “markup men”.

The markup notes would include details such as what font to use, the letter size, and whether the text should be in bold or italics.

You may already have some experience with markup.

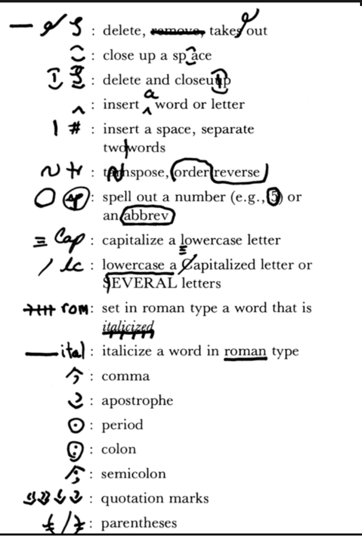
Teacher Markup



"Ralph, Fred, Archie, and Homer: Why television keeps recreating the white male working-class buffoon," Richard Butsch discusses the repeated stereotype of working class males in television portrayed as unintelligent. ~~Also~~ in the article, he mentions a stereotype regarding middle class families. ~~Instead of~~ the husband acting like the "buffoon", like in a working class family he acts sensible and mature, and the wife acts unintelligently. ~~However~~, the only example Butsch provides is the sitcom "I Love Lucy" which aired in 1950's. Today, sitcoms are not nearly as

When your teacher does this to your essay, he or she is "marking up" your essay with notes for improvement.

Proofreading Marks



: delete, ~~remove,~~ takes out
: close up a sp ace
: delete and closeup
: insert word or letter
: insert a space, separate twowords
: transpose, order reverse
: spell out a number (e.g., 5) or an abbrev
: capitalize a lowercase letter
: lowercase a Capitalized letter or SEVERAL letters
: set in roman type a word that is *italicized*
: italicize a word in roman type
: comma
: apostrophe
: period
: colon
: semicolon
: quotation marks
: parentheses

Over time, certain proofreading marks are shortened and standardized. So instead of writing full sentences describing what your teacher would like to see, he or she will write little symbols letting you know that you need to add a missing comma or split a single paragraph into two paragraphs.

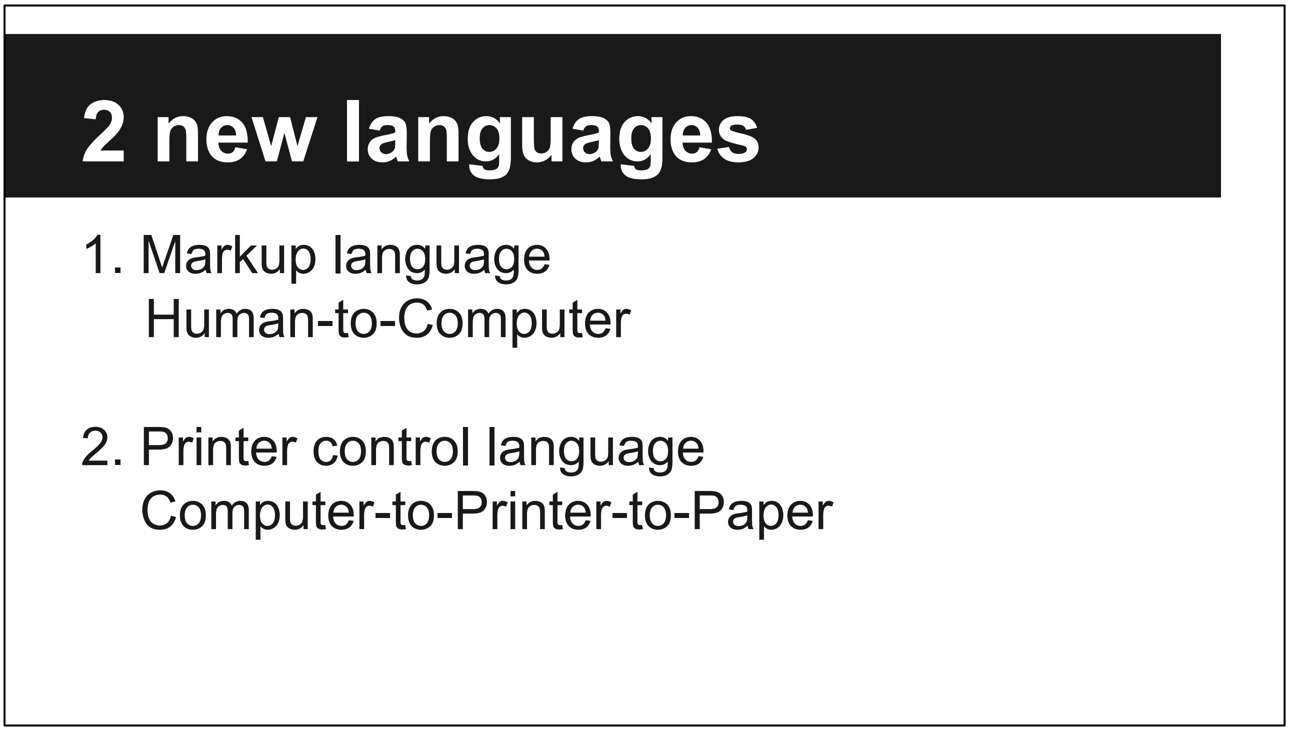
These shorthand symbols define a “markup language”.

The Digital Age

Markup languages had been in existence long before computers were ever imagined.

When computers arrived in the mid 20th century, publishers transitioned away from hand-written notes and letter-by-letter typesetting. Instead of rendering text into an inky panel of lettered metal blocks, they rendered documents into digital images, which were then sent to a printing device.

But the overall publishing workflow didn't change. It just transformed. Instead of writing instructions for a human typesetter to follow, we now write instructions for a computer to follow. We give the computer some text and give it instructions about how we'd like that text to appear: centered, all-caps, and bold, for example.



2 new languages

1. Markup language
   Human-to-Computer

2. Printer control language
   Computer-to-Printer-to-Paper

The transition to digital publishing required the creation of 2 new types of computer languages.

The first, the **markup language**, allowed humans to tell computers how to lay out a page of text. Human -> Computer

The second, the **printer control language**, allowed computers to tell printers how to print a digital image onto paper. Computer -> Printer -> Paper

We don’t teach printer languages (e.g. PostScript) much anymore. In the 1960s every printer manufacturer invented their own computer language for interacting with their particular brand of printer. This was very annoying. You had to write new code for every new device. So scientists standardized the hardware interface.

That means we get to focus on writing markup language and someone else writes the code to transform that markup language into the appropriate printer control language for your specific printer.

B.H. (Before HTML)

- GenCode & GML
- SGML
- TeX

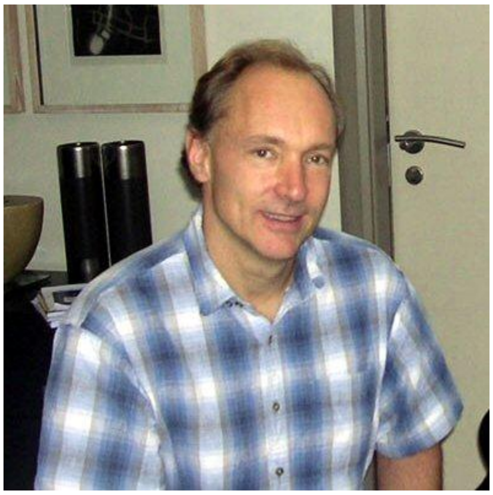
Before HTML there were many attempts to create a standard computer publishing markup language.

First came William Tunnicliffe’s GenCode and Charles Goldfarb’s GML in the 1970s.

Goldfarb later combined GenCode and GML into SGML in 1974, and collaborated with Tunnicliffe on making it an international standard.

Donald Knuth created another markup language called TeX in the 70s and 80s that became (and remains to this day) popular for printing academic and scientific journals. This is why so many academic papers look exactly the same.

Tim Berners Lee



The one markup language that really took off was invented by this nerd: Sir Timothy John "Tim" Berners-Lee, a British computer scientist

Not only did he invent HTML, he also invented the World Wide Web. It was a package deal.

TimBL created the world's first HTTP client and server. (Do we all remember what that means?) He made the first successful HTTP request in Nov. 1989.

He created the world's first website: http://info.cern.ch/ in 1991.

And, inspired by SGML, he wrote that website in a new markup language he invented called HTML. In fact, Tim considers HTML itself to be an SGML application.

HTML is now the main markup language for creating web pages and is likely the most used markup language in the world today.

This is for everyone!

Tim is British. The English are very proud of him. So proud, in fact, they they honored him during the 2012 Summer Olympics opening ceremony. In the middle of London Olympic Stadium, he used an old computer to tweet “This is for everyone.” The message was spelled out in lights across the 80,000-strong crowd.

The computer he tweeted from was the same model he used to create the web in 1989, a vintage NeXT. I don’t know how he got Twitter to work on that old thing.

In 2004, Berners-Lee was knighted by Queen Elizabeth II for his pioneering work. (In contrast, Turing never received knighthood) http://news.bbc.co.uk/2/hi/8249792.stm

“This is for everyone” would serve to be a guiding principle for Tim and the World Wide Web as a whole.

W3C

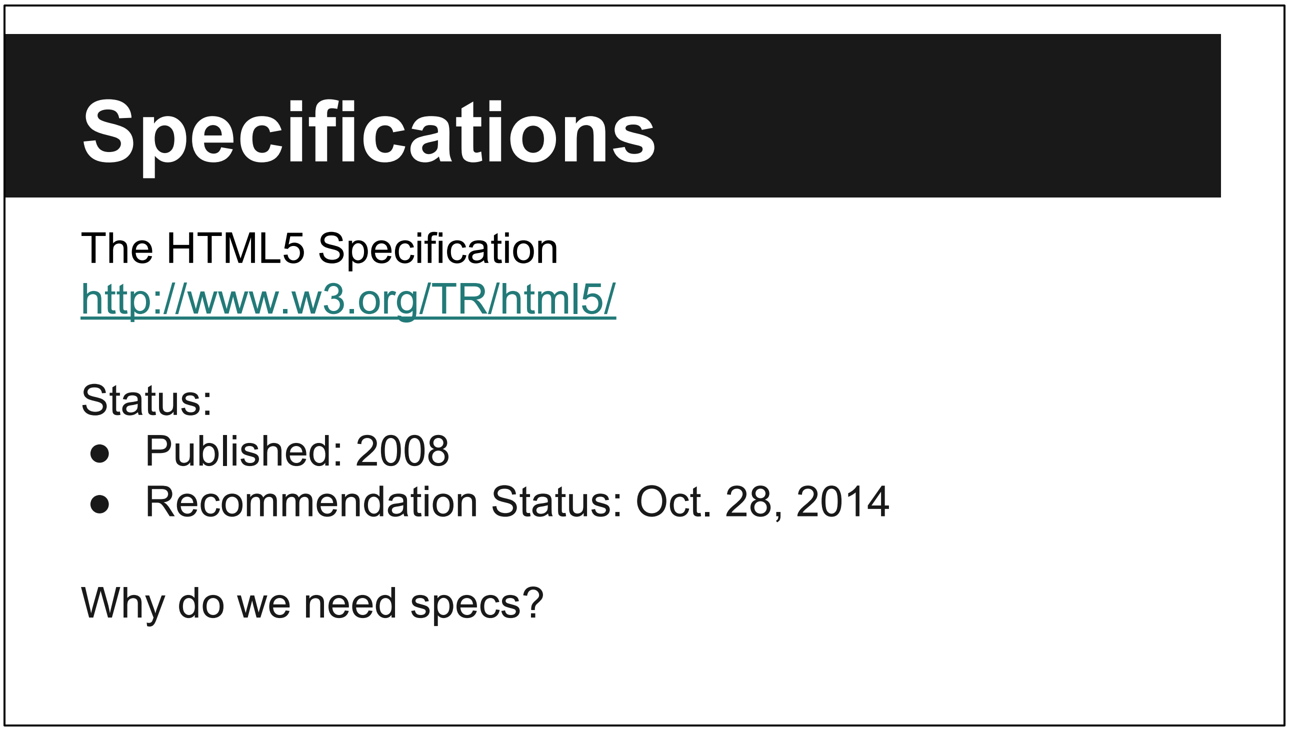
The World Wide Web Consortium (W3C, www.w3.org) — directed by the inventor of the Web and HTML, Tim Berners-Lee — is the organization responsible for shepherding the development of Web standards.

Berners-Lee is the founder and director of the World Wide Web Consortium (W3C), which oversees the Web's continued development.

He’s also a member of the MIT community, holding the Founders Chair at the MIT Computer Science and AI Lab (CSAIL) and advising the MIT Center for Collective Intelligence.

The web standards community doesn’t work the same way the Ruby community does. In most programming languages and large projects (like Rails), when something needs to be updated, a core group of contributors makes the changes. Contributions are accepted from anyone and the code is open source, but the direction of the product is directed by an inner circle.

The W3C standards process is different.



Specifications

The HTML5 Specification
http://www.w3.org/TR/html5/

Status:
- Published: 2008
- Recommendation Status: Oct. 28, 2014

Why do we need specs?

There isn't an HTML "language" per-se. I can't download and install an `html` command that will read my HTML code and produce documents for me, the way I would with Ruby (or most other languages).

Instead, the W3C publishes a document called a "specification" (or "spec", for short). This document defines the syntax and features of HTML.

After a long discussion period, the spec advances through a series of pre-defined stages ending with "Recommendation Status". HTML version 5 reached this final stage just last year.

At "recommendation status", browser makers (Chrome, Firefox, Internet Explorer, Safari, Opera) are encouraged to adopt the new language standards by updating their browsers. Only after widespread adoption should web developers feel comfortable using the newly introduced language features.

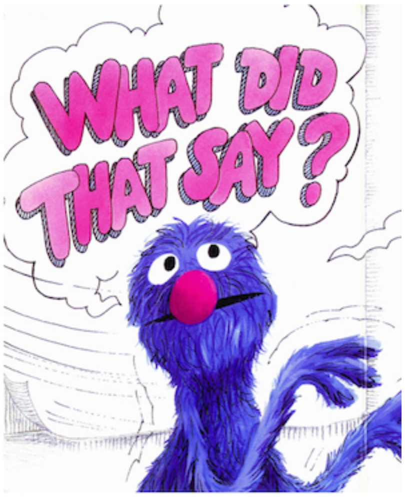
Why does it work this way? Why do we need specs?

We need a universally agreed upon standard to prevent inconsistency in how web pages are rendered. Your job would be a lot harder if you had to write custom code for every browser you wanted to support. It's the PCL problem all over again.

The downside, however, is that the W3C has no power to force browser makers to obey the standard. So there still remain issues with HTML code that displays differently across different browsers. But it’s getting better. We’ll see some some specific examples later of things that aren’t supported across all browsers, particularly when we get to Rails.

Web languages like HTML, CSS, and JavaScript live in this weird world where they write documents about how things should work, even writing sample code for how it would work, but don’t actually write or distribute the programs that run that code. (We’re covering this topic here in the “easy” HTML lecture so we don’t have to do it again later in CSS and JS).

HTML5



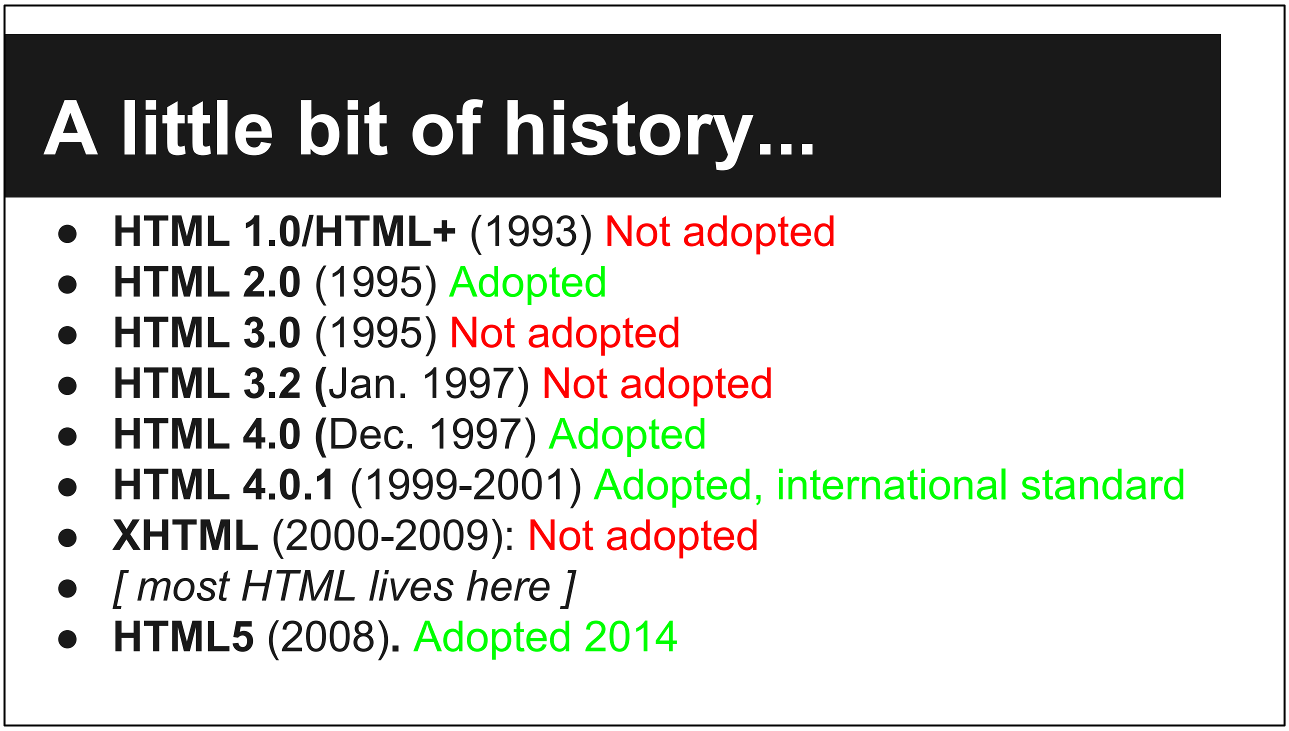
HTML version 5 was released in 2014?!

So does that mean HTML5 was just released last year? Because I've been hearing chatter about it for much longer than that.

That's true. We've been talking about HTML5 for a long time. It can literally take years for the standardization process to advance to "recommendation status".

Thankfully, a side-effect on the "spec" process is that browser makers routinely "jump the gun". They start implementing new language features before they reach recommendation status. So browsers have supported many HTML5 features for a while now.

Homework: What is the risk of using HTML/CSS/JS features that haven't reached recommendation status?



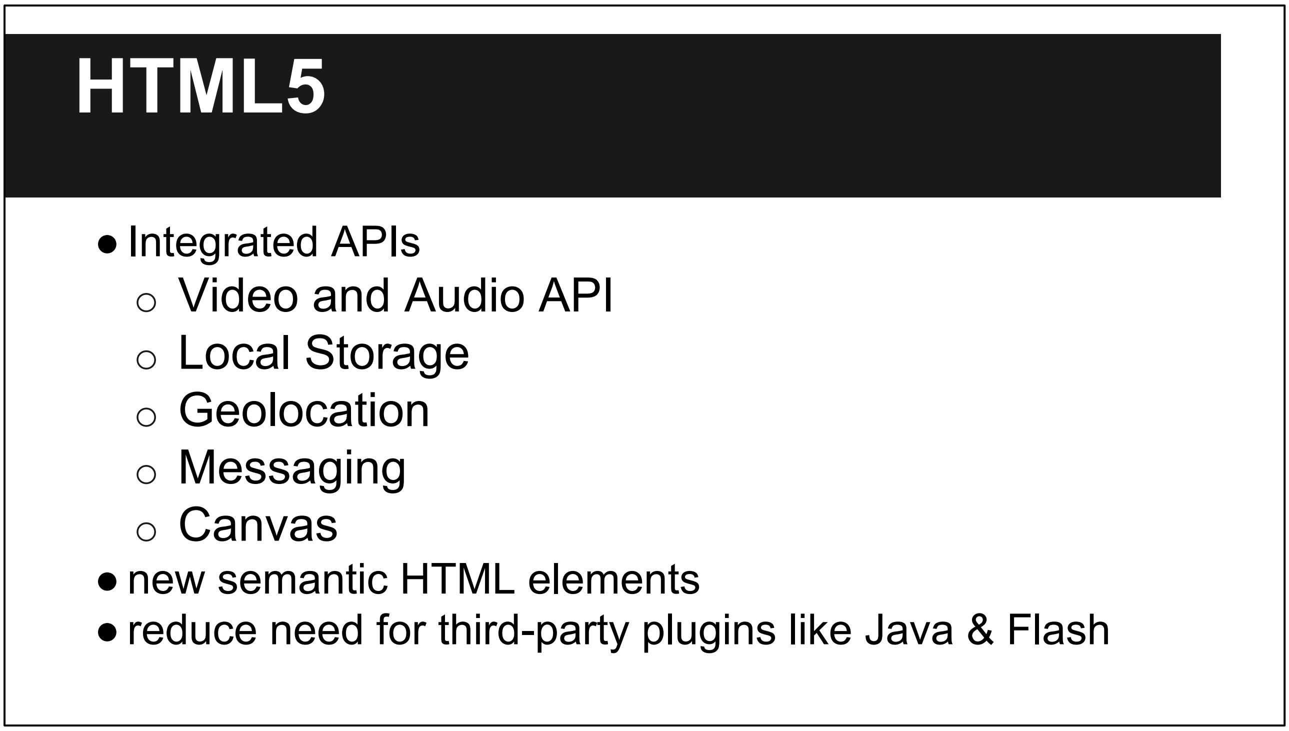
A little bit of history...

- **HTML 1.0/HTML+** (1993) Not adopted
- **HTML 2.0** (1995) Adopted
- **HTML 3.0** (1995) Not adopted
- **HTML 3.2 (**Jan. 1997) Not adopted
- **HTML 4.0 (**Dec. 1997) Adopted
- **HTML 4.0.1** (1999-2001) Adopted, international standard
- **XHTML** (2000-2009): Not adopted
- *[most HTML lives here]*
- **HTML5** (2008)**.** Adopted 2014

This list illustrates the evolution of HTML over the past couple of decades. We won't go into much detail. Just note a few things.

1. There are dead ends. Lots of them. Even the experts make mistakes and need to start over again sometimes. The 2000s were a lost decade. JS has the same problem.
2. Most HTML code (and HTML writers) are stuck at HTML4 because it was the "only game in town" for such a long time (1997-2008/14). Rails still prefers to think in HTML4.

p.s. http://evolutionofweb.appspot.com/ (Skip this for time)



HTML5

- Integrated APIs
  - Video and Audio API
  - Local Storage
  - Geolocation
  - Messaging
  - Canvas
- new semantic HTML elements
- reduce need for third-party plugins like Java & Flash

HTML5 introduced a lot of cool features to HTML.

We finally have a way to play Video and Audio without resorting to plugins like Flash and Quicktime.

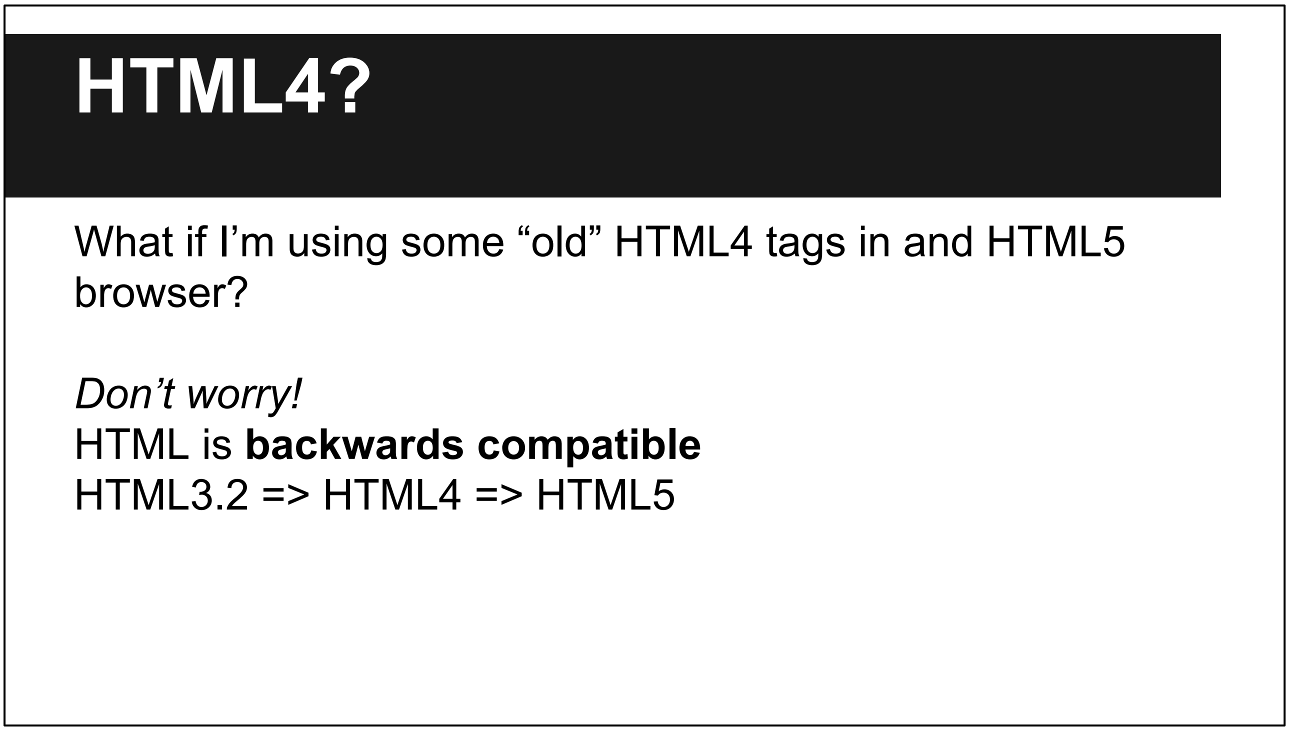
Geolocation means browsers can ask you where you’re located.

Canvas means you can paint whatever you want into a webpage, just like Flash used to do.

Messaging means you can do real-time interaction like chat and online gaming.

There are also new Semantic HTML elements that we’ll talk about later.

These changes collectively killed Java Applets and Flash for many - but not all - use cases.



HTML4?

What if I’m using some “old” HTML4 tags in and HTML5 browser?

Don’t worry!
HTML is **backwards compatible**
HTML3.2 => HTML4 => HTML5

But, like I said, most of the web still uses HTML4. So, is that a problem?

It’s ok. The HTML specification writers make sure that old HTML continues to work in new browsers.

http://www2.warnerbros.com/spacejam/movie/jam.htm

HTML remains backwards compatible. Older versions are compatible with newer versions.

XHTML... WTF

HTML doesn’t have errors!

```
<html><is>awesome<!>...
```

The HTML community was stuck for most of the 2000s working on a project that went nowhere. Why?

Browser implementations of HTML all have an interesting bug/feature… they don’t throw errors!

This is “valid” HTML code. Your browser won’t complain. (Demo this.)

There’s no such thing as an HTML “syntax error”. Browser makers take whatever garbage you give them and try their best to make sense of it.

XHTML... WTF

XHTML introduced HTML syntax errors

Pro
Faster browsers.

Con
I don’t want to update my old, broken code.

A variation of HTML called XHTML 1.0 was published in 2000. The various standards organizations spent much of the 2000s perfecting it, before abandoning it in 2009 in favor of HTML5.

XHTML combines HTML with it’s much stricter cousin XML. If you wrote bad HTML, the page errored by going blank. Just a plain white screen.

The W3C thought this would make the web faster. Browser makers could remove all the code that tries to understand junky HTML.

HTML writers, on the other hand, didn’t care for updating their old, bad HTML code. Websites lived online for a surprisingly long time.
http://www2.warnerbros.com/spacejam/movie/jam.htm

And they didn’t want their users to use browsers that caused their old sites to break.

So everyone ignored XHTML.

"The attempt to get the world to switch to XML … all at once didn't work. The large HTML-generating public did not move“ -Tim Berners Lee in 2006.

So browsers are slower, but developers are happier. And you never get to experience the joy of an HTML syntax error.

Validating HTML

W3C Markup Validation Service
http://validator.w3.org/

That doesn't mean there's no such thing as an HTML syntax error. It's just that browsers don't show them. They just roll right through them, sometimes producing unexpected results, some of which are inconsistent between browser vendors.

The W3C provides a convenient HTML validation service to check for errors. Use this to debug your HTML. Because that is a thing that you still need to do.

If you make a mistake, but it still looks ok in Chrome, it might not look ok in IE. But if you write valid HTML, you're more likely to have cross-browser compatibility (more likely, but no guarantee).

Besides, this will help with your homework.

Examples of HTML "bugs":

- Unclosed/Mismatched tags.
- Misspelled, invalid, or broken attributes.
- Missing required attributes.
- Missing or extra quotes.
- Improper nesting.

HTML Feature Support

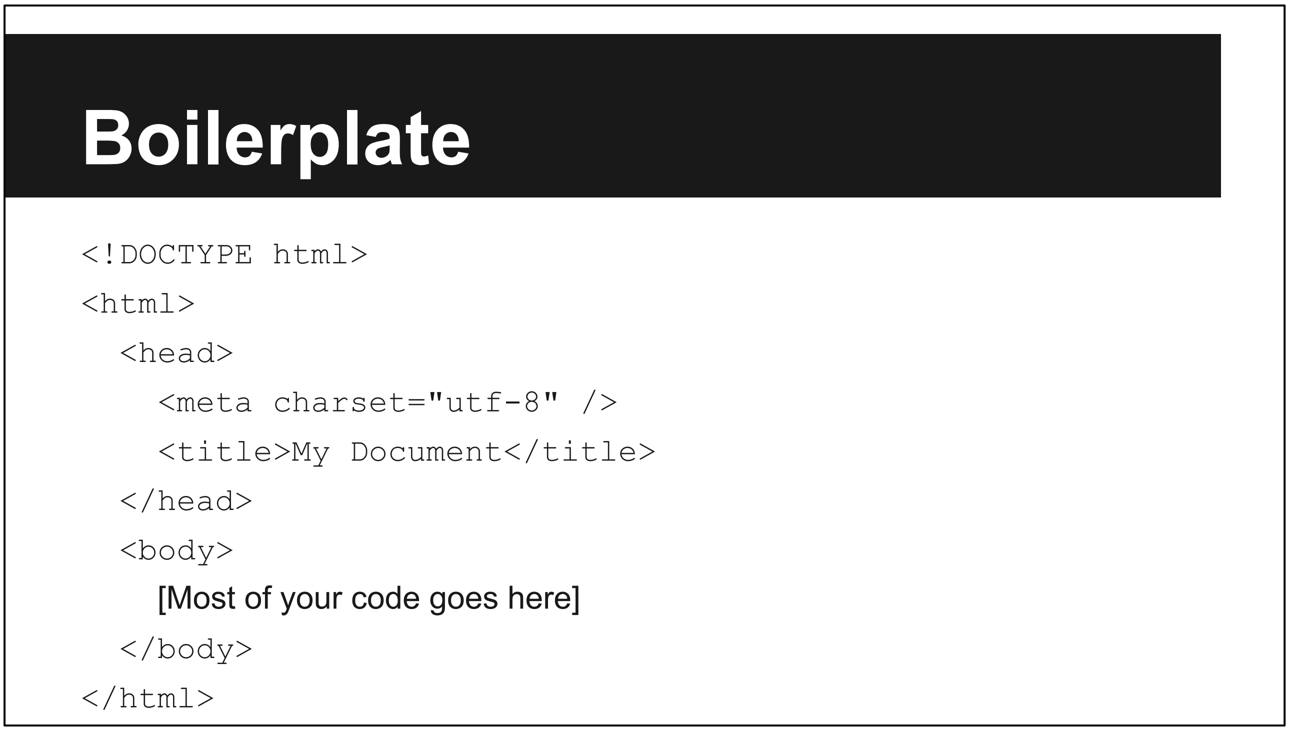
What can I use now?
What level of support do browsers offer?

Feature support is changing rapidly:
caniuse.com

HTML continues to advance and it can be hard to keep track of what did and didn't make it into a particular spec. Or, worse yet, what browser vendors did or didn't try to do (and how).

Sites like Can-I-Use will give you great summaries of which browsers support which features (HTML, CSS, and JS). So when you read about something new, you can quickly check for browser support and quirks there, without having to know the spec inside-and-out.

This will also be useful for your homework.



Boilerplate

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>My Document</title>
  </head>
  <body>
    [Most of your code goes here]
  </body>
</html>
```

Boilerplate, noun, standardized pieces of text for use as clauses in contracts or as part of a computer program.

HTML has some standard boilerplate that must be included in every document.

Ruby just let’s you write code. HTML require this leading and trailing code just to get started.

Remember, though, there are no syntax errors, so it’s technically not required. But you won’t validate without it.

Note that the whitespace here doesn’t matter. Both vertical *and horizontal* whitespace is purely for human readability. If all the HTML were on a single line, the browser would be able to handle that just fine. But since you’re a computer, you should use whitespace to help you understand your code.

Also note that capitalization doesn’t matter. Tags used to be in all-caps, but since we can get away with lowercase and that’s less keypresses, that’s the convention.

tip: Sublime will give you lots of this in an HTML document if type html<tab>

Doctype

HTML4

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

HTML5

```
<!DOCTYPE HTML>
```

The first line of any HTML document is the DOCTYPE. This tells the browser what version of HTML to expect. Browser will just guess if you don’t include. And they could guess incorrectly.

Charset

HTML4

```
<meta http-equiv="Content-Type"
content="text/html; charset=utf-8">
```

HTML5

```
<meta charset="UTF-8">
```

The next relevant line of the boilerplate is the character set declaration. This tells the browser whether the characters in the page are in ASCII or UTF-8. Without it, the browser will just guess. If it guesses wrong, you could be introducing a security problem.

https://code.google.com/p/doctype-mirror/wiki/MetaCharsetAttribute

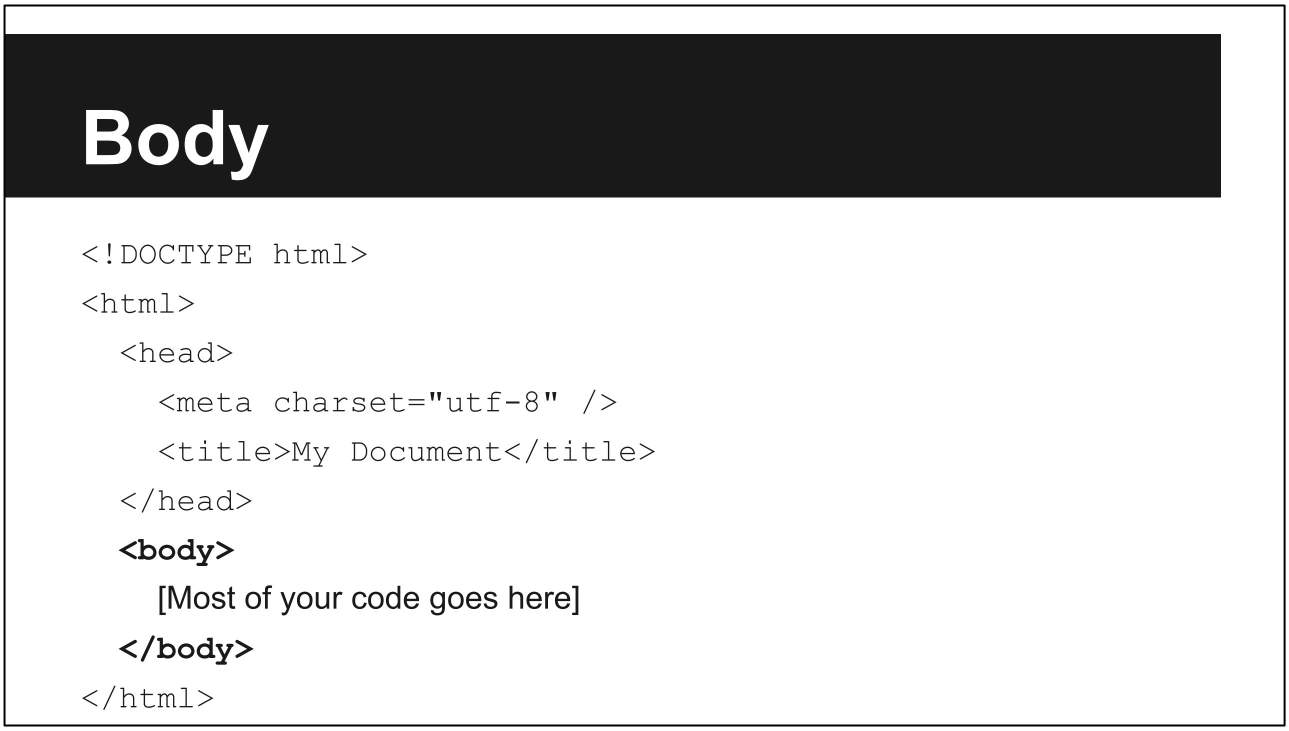
You should always specify UTF-8. Some early web pages default to ASCII, but you'll have problem with £ symbols and such un-American nonsense.

80% of the web is on UTF-8 now. http://w3techs.com/diagram/history_technology/en-utf8

Title

```
<title>My Document</title>
```

The <title> element defines the text that shows up in the browser tab and on bookmarks. It’s required by the spec, so it’s a validation error if you skip it.



Body

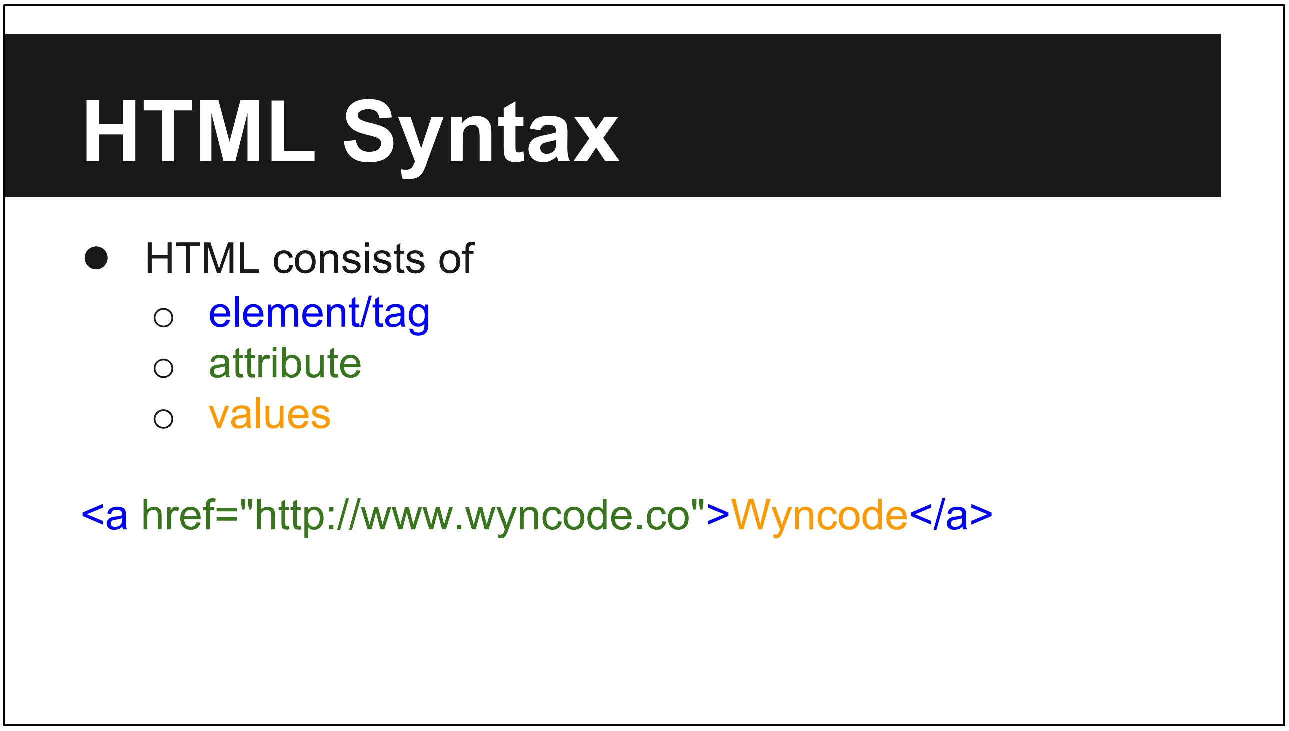
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>My Document</title>
  </head>
  <body>
    [Most of your code goes here]
  </body>
</html>
```

The rest of your code goes in the <body> tag. Sometimes you'll put stuff in the <head>, but mostly you're working within <body>.

Think of this like an HTTP request/response.
- doctype == status line
- head == headers
- body == body

Good designs pop up again and again.

We won't be covering every HTML tag. Just like we didn't cover every Ruby Class and Method. There just isn't enough time and you wouldn't be able to memorize it all from the lecture anyway. Besides, there are only about 100 tags, and a bunch are obsolete, so it's not actually impossible to memorize every tag some day, if you wanted to:
http://webdesignfromscratch.com/html-css/list-of-html-tags-with-semantic-usage/

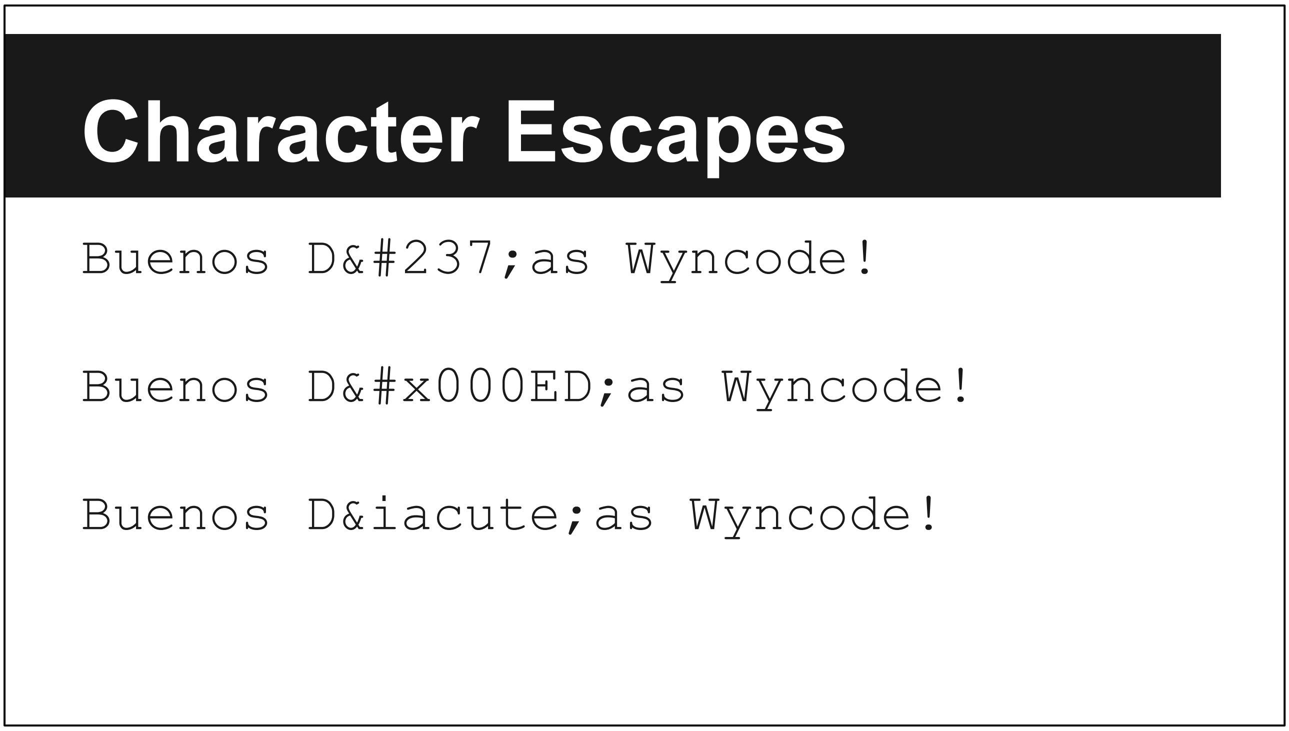


HTML Syntax

- HTML consists of
  - element/tag
  - attribute
  - values

```
<a href="http://www.wyncode.co">Wyncode</a>
```

The standard HTML syntax consists of an element or tag, one or more attributes, and some internal text value.



Character Escapes

```
Buenos D&#237;as Wyncode!

Buenos D&#x000ED;as Wyncode!

Buenos D&iacute;as Wyncode!
```

[enter livecoding mode here]

UTF-8 allows you to enter foreign characters into your documents, but you should avoid putting them in tags and attributes.

Some browsers have trouble with foreign characters, even when you specify the UTF-8 character set. So HTML has an “escape syntax”.

Similar to Ruby Strings, when HTML parsers see the “&” character, they enter a special “mode”. All the characters between the & and the ; are “special”.

While in escape mode, if the browser sees a #, it enters another mode. The remaining characters are treated as a numeric code.

Whether via text or numeric code, character entities are the safest way to include foreign characters into an HTML document.

This will be useful in your homework.

http://dev.w3.org/html5/html-author/charref
http://unicode-table.com/en/
http://www.w3.org/International/questions/qa-escapes

Do I need HTML Entities?

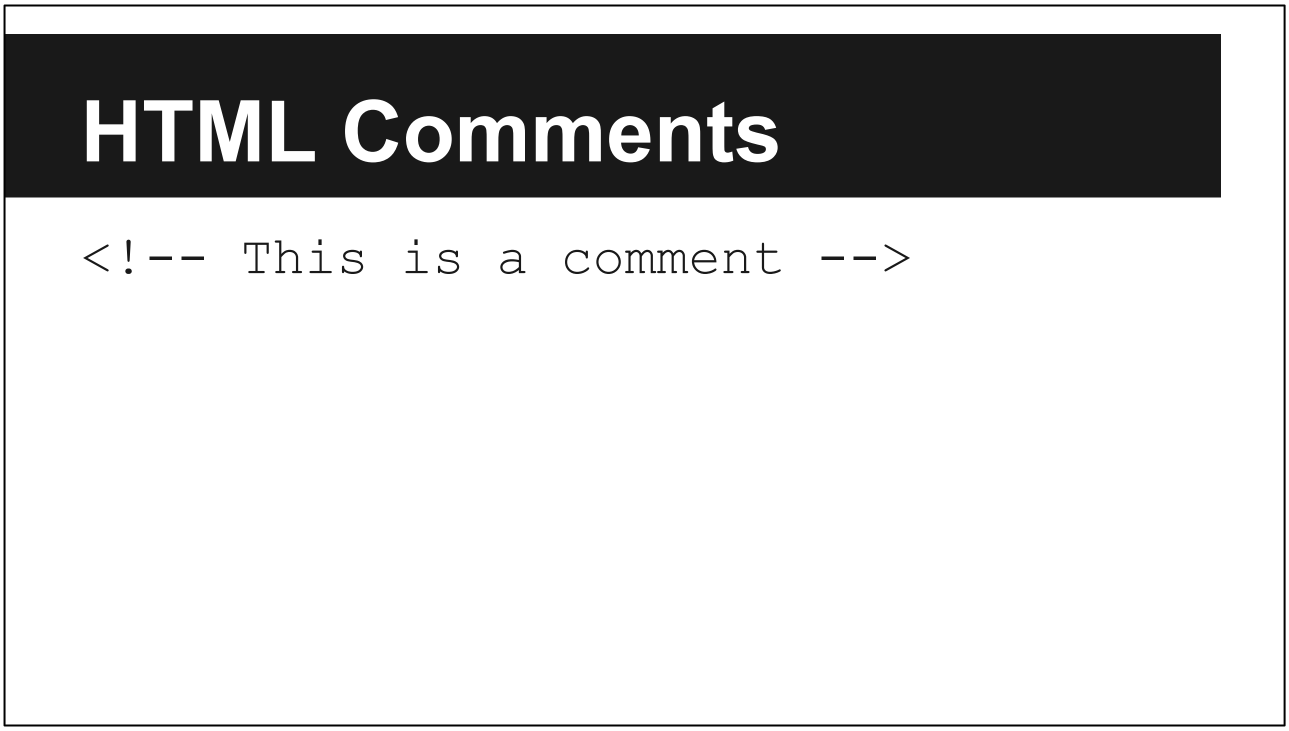
```
&lt;Night&gt; &amp; &lt;Day&gt;
```

<Night> & <Day>

Since most browsers recognize UTF-8 these days, you can technically get away with not using the escape syntax for some foreign characters.

However, you still need to escape characters reserved by the HTML syntax: e.g. less than, greater than, ampersand.

Even if the browser doesn’t understand UTF-8, you can sneak foreign characters in using this escape syntax.



HTML Comments

```
<!-- This is a comment -->
```

HTML has comments. Put whatever you want between <!-- and -->. HTML will ignore it.

<img>

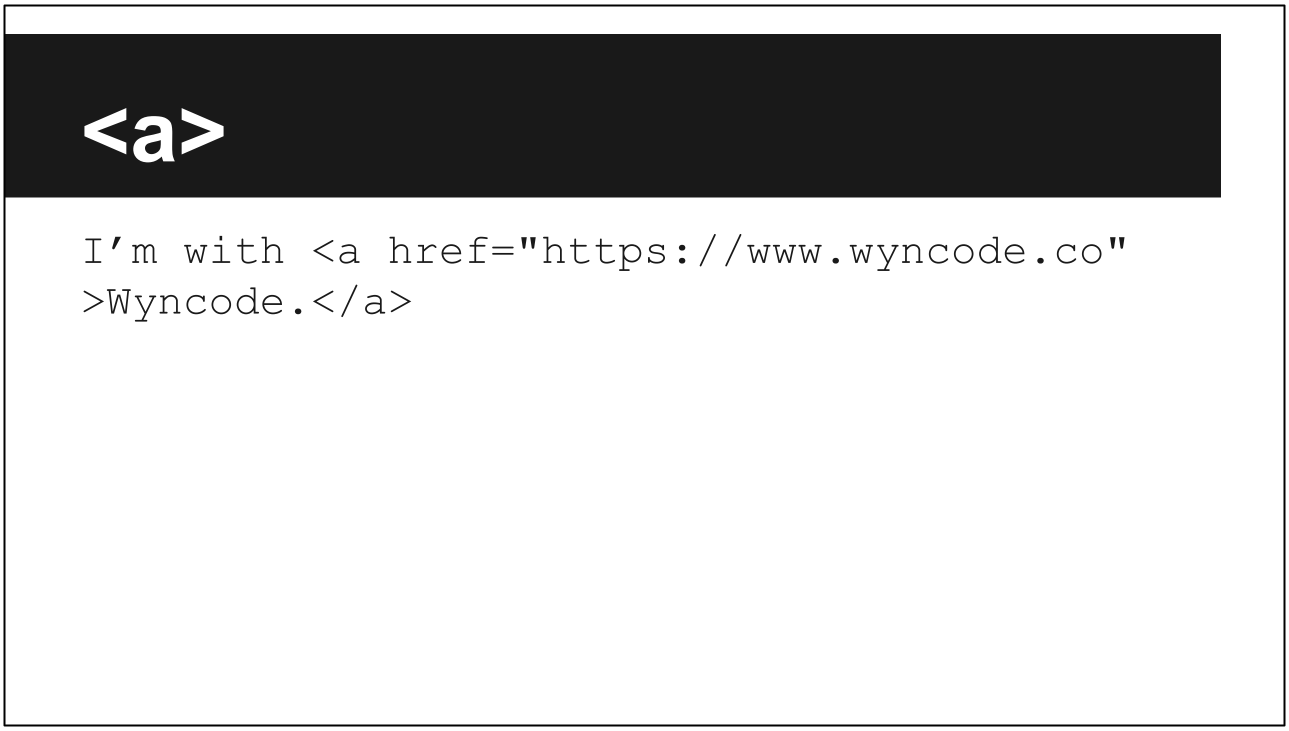
```
<img src="profile.jpg" alt="Profile Pic" />
```

The `alt` attribute is “required”.

This is how you put images into your page. Note that img tags are self-closing. They don’t require a separate </img>. In fact, even the final / is optional.

There are very few self-closing (aka “void”) tags.
http://www.colorglare.com/2014/02/03/to-close-or-not-to-close.html

The `alt` attribute is “required” (again, browsers won’t complain, but the W3C will). The alt text is used if the image is broken or the page is viewed as text-only (i.e. by blind people).



<a>

```
I’m with <a href="https://www.wyncode.co"
>Wyncode.</a>
```

The <a> (anchor) tag links multiple pages together. This is what makes the web the web - a interconnected series of documents.

<input>

```
<input type="text" name="lastname">
```

The <input> tag specifies an input field where the user can enter data.

On it’s own it doesn’t do anything. But when combined with a <form> tag (or some JavaScript), it allows users to send data to your server.

There are multiple options for the type attribute. We’ll talk about some of those more when we get to Rails.

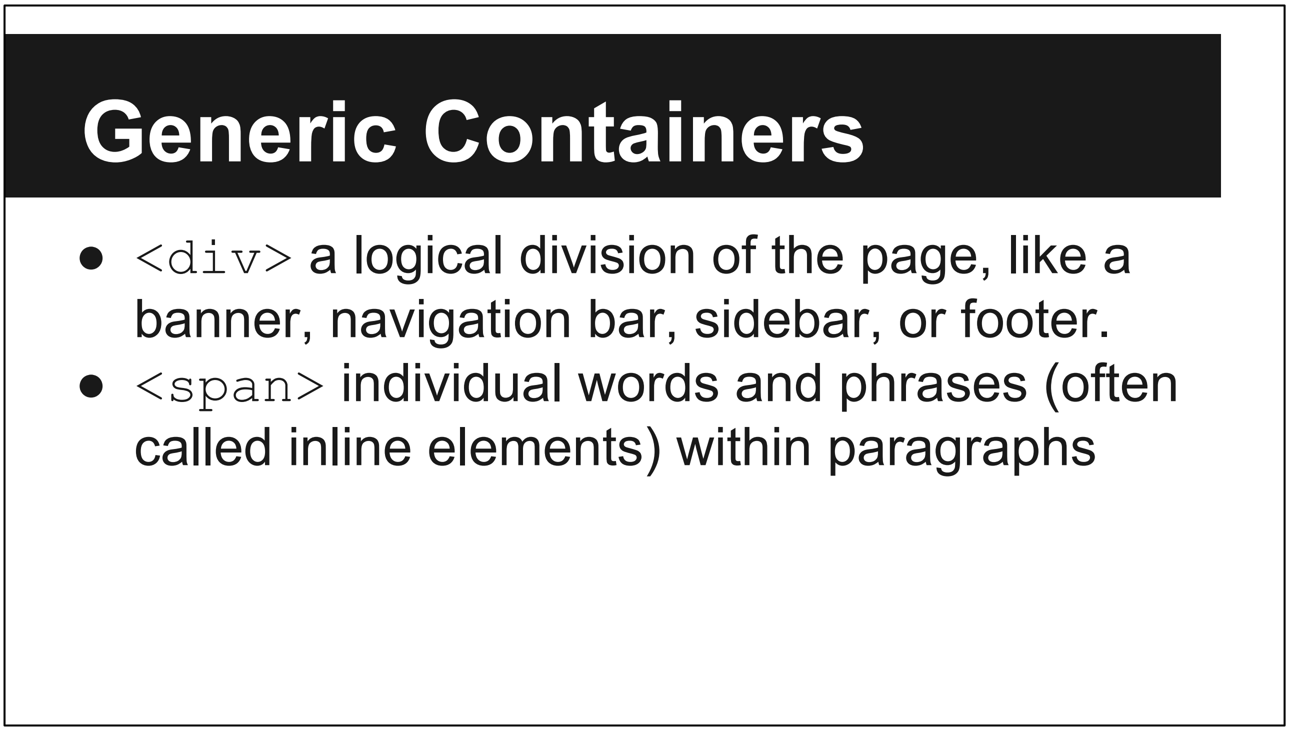
Buttons

```
<input type="button" value="Submit" />

<button type="button">Submit</button>


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

A button can be considered a special type of input tag. But there’s also a separate <button> tag. They do the same thing.

By default, clicking a button doesn’t do anything. You have to add something either in the <button> value or with JavaScript. You can’t add anything to the <input type=” button”> style buttons.

The “submit” is a special type of button input for submitting HTML forms. We’ll talk about forms more later when we get to Rails. In static front-end coding, there’s no point in collecting data if you don’t have a back-end to send it to.



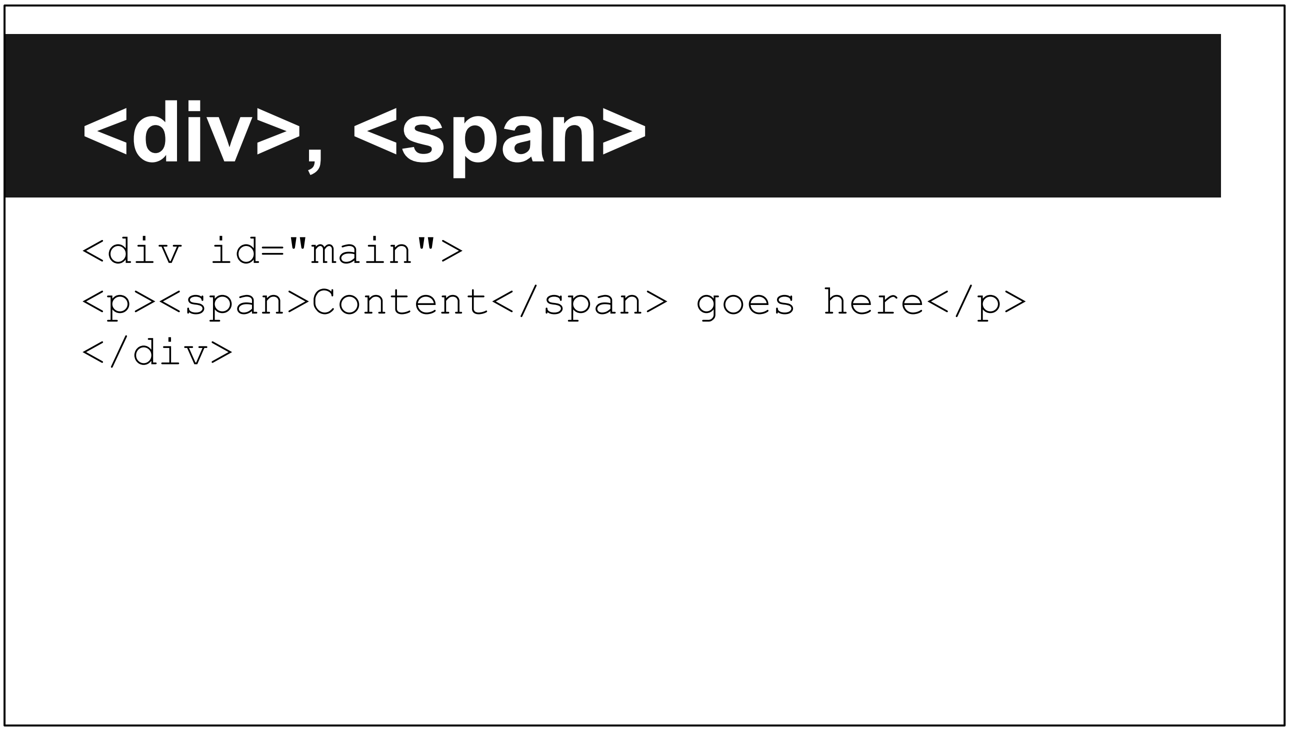
Generic Containers

- `<div>` a logical division of the page, like a banner, navigation bar, sidebar, or footer.
- `<span>` individual words and phrases (often called inline elements) within paragraphs

Just like Ruby has containers (Array and Hash), HTML has 2 general purpose element containers: <div> and <span>.

HTML5 introduced a number of semantically significant containers like <header>, <footer>, <article>, and <section>. But there were already some semantic tags in older versions of HTML, like <h1>..<h6> and <p>. Despite all the variation, there are only two behaviors.

<div>-style tags include a linebreak (i.e. they are “blocks”). <span> tags don’t (i.e. they are “inline”).

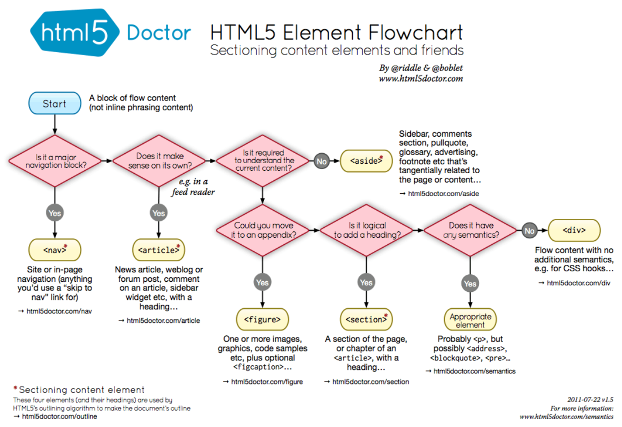


<div>, <span>

```
<div id="main">
<p><span>Content</span> goes here</p>
</div>
```

Here’s an example.

HTML5 Semantic Elements


html5 Doctor
HTML5 Element Flowchart
Sectioning content elements and friends
By @riddle & @boblet
www.html5doctor.com
Start
A block of flow content (not inline phrasing content)
Is it a major navigation block?
Yes
<nav>*
Site or in-page navigation (anything you'd use a "skip to nav" link for)
→ html5doctor.com/nav
Does it make sense on its own?
e.g. in a feed reader
Yes
<article>*
News article, weblog or forum post, comment on an article, sidebar widget etc, with a heading…
→ html5doctor.com/article
Is it required to understand the current content?
No
<aside>*
Sidebar, comments section, pullquote, glossary, advertising, footnote etc that's tangentially related to the page or content…
→ html5doctor.com/aside
Could you move it to an appendix?
Yes
<figure>
One or more images, graphics, code samples etc, plus optional <figcaption>…
→ html5doctor.com/figure
Is it logical to add a heading?
Yes
<section>*
A section of the page, or chapter of an <article>, with a heading…
→ html5doctor.com/section
Does it have any semantics?
Yes
Appropriate element
Probably <p>, but possibly <address>, <blockquote>, <pre>…
→ html5doctor.com/semantics
No
<div>
Flow content with no additional semantics, e.g. for CSS hooks…
→ html5doctor.com/div
* Sectioning content element
These four elements (and their headings) are used by HTML5's outlining algorithm to make the document's outline
→ html5doctor.com/outline
2011-07-22 v1.5
For more information:
www.html5doctor.com/semantics

http://html5doctor.com/downloads/h5d-sectioning-flowchart.png

We won't cover all the new HTML5 semantic elements, but this flowcharts will help you figure out block-style ("sectioning") element is most appropriate for your code.

HTML nesting

When elements contain other elements, each element must be properly nested, that is, fully contained within its parent.

```
<div><p><a href="#">Link</p></a>
```

Elements placed within containers must be properly nested.

Open tag1, open tag2, close tag1, close tag2.

This sample code has some nesting problems. The closing </div> tag is missing. And the <p> and <a> tags “overlap”.

Proper use of whitespace can help you identify these nesting errors.

IECCs

Getting IE8 to understand HTML5:

```
<!--[if lt IE 9]>
<script src="//html5shiv.
googlecode.com/
svn/trunk/html5.js"></script>
<![endif]-->
```

While HTML is just a markup language, it does contain limited support for one form of control flow. This is what an if-statement looks like in HTML.

This isn’t actually part of the HTML spec. As you can see, it’s a comment. But IE (and only IE) recognizes this comment as special. It allows you to add pieces of HTML to your document *if* the browser is IE (or some particular version of IE). This is particularly useful for attaching messages or scripts to a page that *fix* IE bugs.

This is an “Internet Explorer conditional comment” (IECC for short) targeting IE 9 and earlier (lt means “less than”). In particular, this code will help older versions of IE accept HTML5 tags without choking.

Again, not a syntax error. Just weird behavior.

Accessibility

Accessibility Guidelines
http://www.w3.org/TR/WAI-WEBCONTENT/full-checklist.html

Making HTML accessible means making it usable by people with disabilities. Those disabilities range from blindness to the inability to use a keyboard or mouse.

Many government websites have high accessibility requirements, which can make web design for them a bit more difficult. We won’t be going into this topic.

Exploring & Debugging

- View Source
- Chrome Developer Tools
- http://codepen.io/

Don’t forget to validate!

TimBL is fond of saying the web is “for everyone”. It’s a wide open platform. You can see the source code of anything you see online. Take advantage. Click on “view source” to learn how interesting things are made.

To tinker (or to make your work iterations faster), take advantage of Chrome Developer Tools ability to “live edit” HTML code. You can’t save your work, but you can experiment until you find something that looks right, then copy the answer into your code.

[demo this]

CodePen.io is a good HTML/CSS/JS “scratchpad” - a place where you can try out snippets of front-end code to see what they do before adding them to your site. You can even save your best “pens” for later reference and browser through other saved pens for inspiration.

As always, validate your code with the W3C (or some other HTML validator) before you’re done. Just because the browser *looks* correct doesn’t mean the code is syntactically valid.

Semantic HTML

- HTML markup describes the *meaning* of the content, that is, the semantics.
- HTML markup does not define *appearance* of the content; that’s the role of CSS (Cascading Style Sheets).

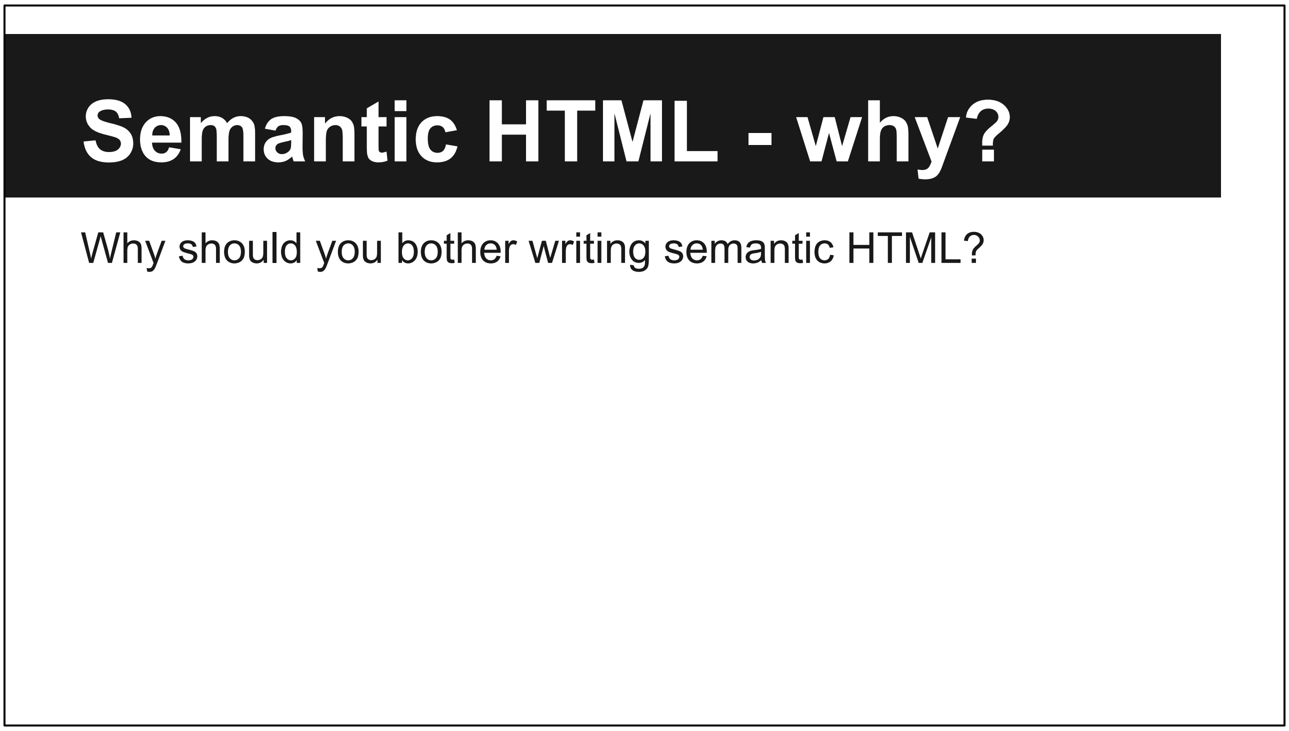
If HTML doesn't define appearance, why is the <h1> tag bigger and bolder? Isn't that a contradiction?

Semantic HTML

- Some tags have a different appearance because every Web browser has a built-in CSS file (a style sheet) that dictates how each HTML element displays by default, unless you create your own that overwrites it.
- The default presentation varies slightly from browser to browser, but on the whole it is fairly consistent.
- When creating web pages, choose the elements that best describes the meaning of your content **without regard for their presentation.**

Semantic HTML

- Semantic HTML: <h1>Welcome to PetsMartOnline.com!</h1>
- Non-Semantic HTML: <p>Welcome to PetsMartOnline.com!</p>



Semantic HTML - why?

Why should you bother writing semantic HTML?

- Improved accessibility and interoperability (content is available to assistive technologies for visitors with disabilities, and to browsers on desktop, mobile, tablet, and other devices alike)
- Improved search engine optimization (SEO)
- (Typically) lighter code and faster pages
- Easier code maintenance and styling

Accessibility is the practice of making your content available to all users, regardless of their capabilities (see www.w3.org/standards/webdesign/accessibility).