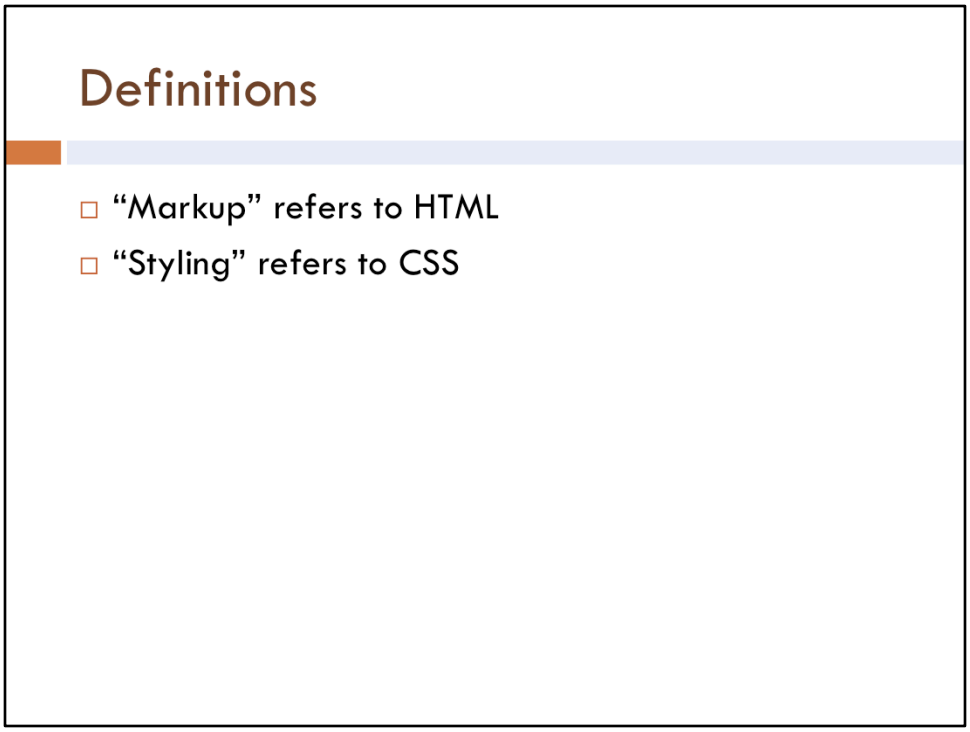
Before we begin: throughout this presentation, I've included various links and citations. They are all required reading. I will be sending out the presentation and notes afterward; please study them at your leisure.



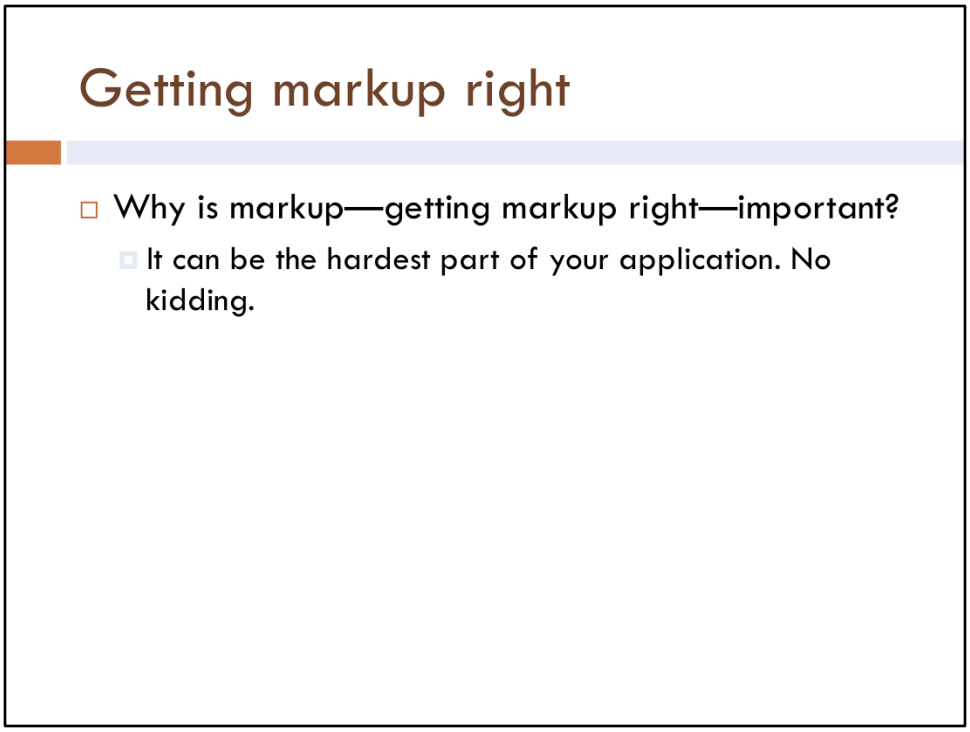
Generally speaking, when I say "markup," I mean HTML, and when I say "styling," I mean CSS. I will also on occasion use the term markup to refer to HTML, CSS and JavaScript together; i.e. your UI implementation.

Getting it right

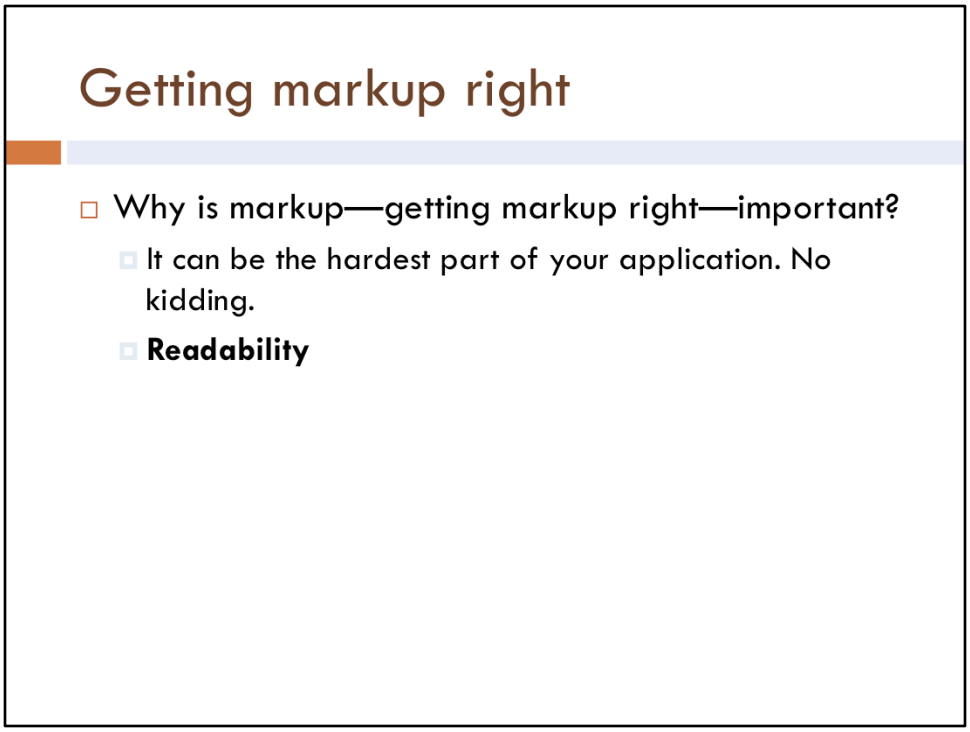
Why good markup is important, and how to write it.

Getting markup right

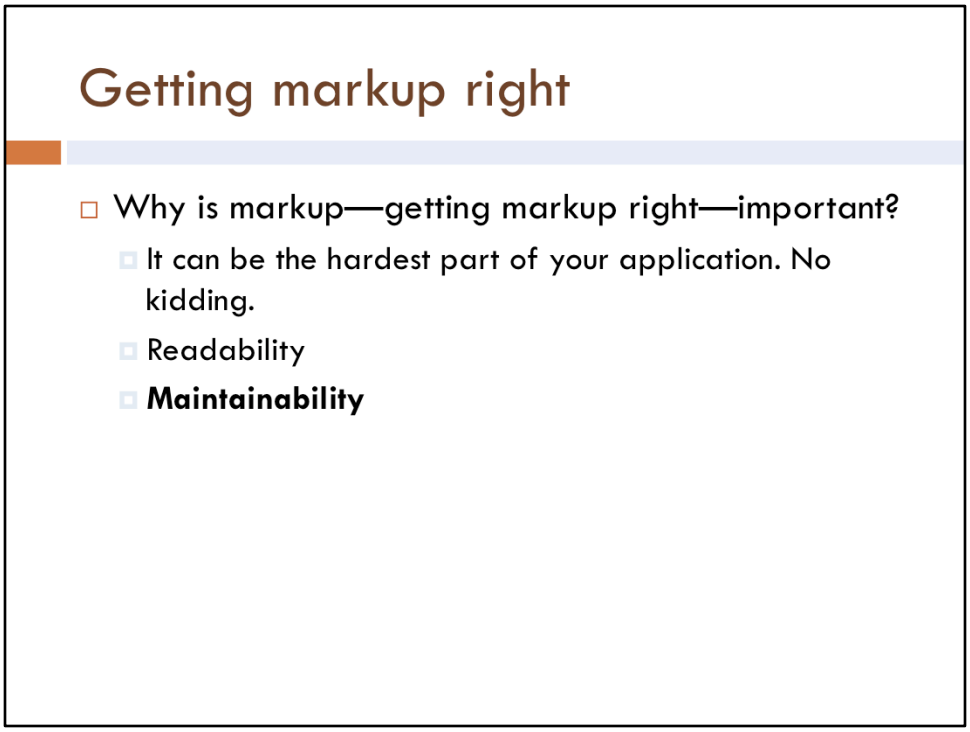
- Why is markup—getting markup right—important?



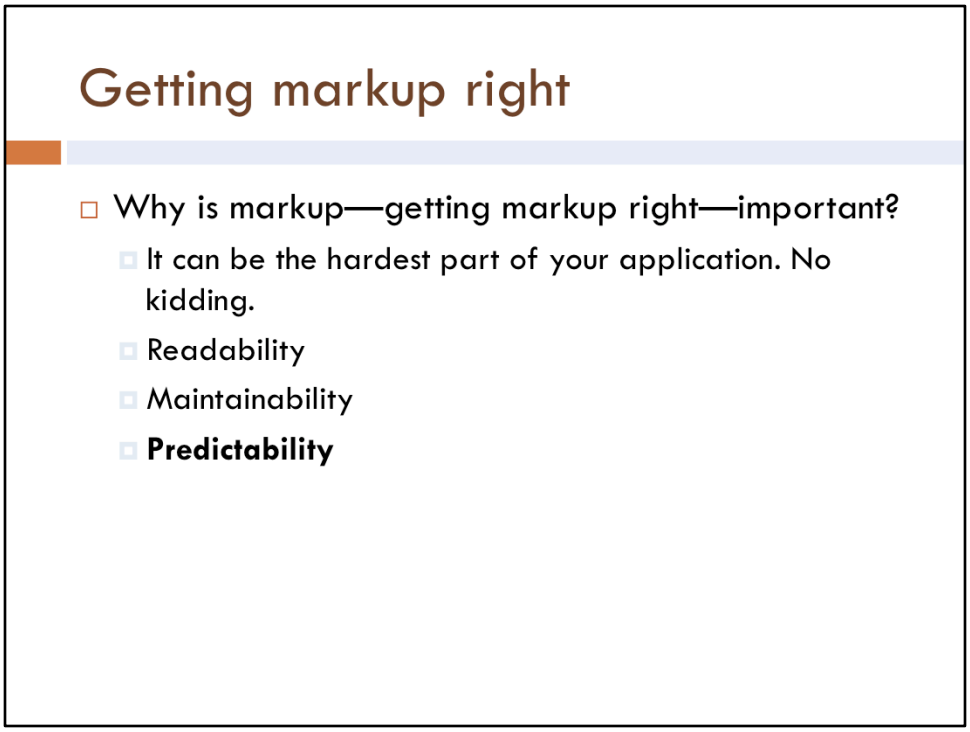
- Markup is harder than you think.
  - You know the importance of good UI design. It doesn't matter how elegant or functional your back-end implementation is; if you don't have something that's easy for your users to learn and use, you have nothing. And the markup is your first line of defense. It's the man behind the curtain.
  - Writing markup is a significant portion of making the dream (the UI design) a reality. You have to make sure it works smoothly, consistently, and in every environment your users even consider. No exceptions. The more complex your application becomes, the harder markup becomes; markup in and of itself is a complex machine with many moving parts, and you've got to keep all those parts in mind, all the time.



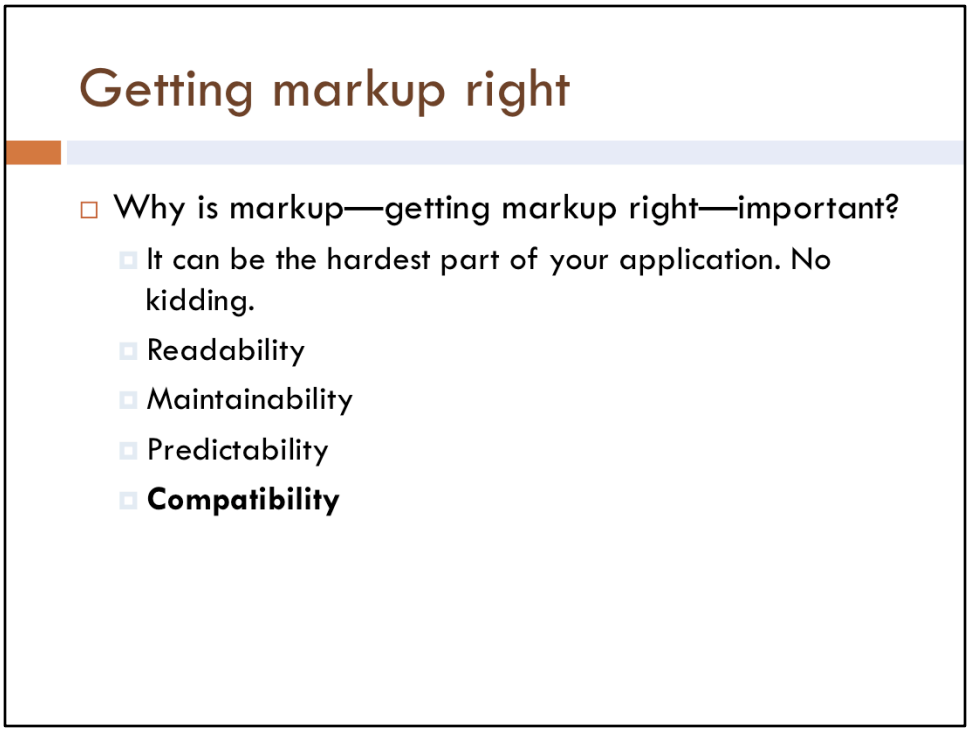
- Readability
  - Markup can suffer just as much from the “spaghetti code” phenomenon as code written in more traditional programming languages, and can, especially when CSS and JavaScript get involved, in fact be harder to detangle than a pot of congealed spaghetti.
  - Follow coding and style guidelines; well-commented, well organized markup will make your life easier.



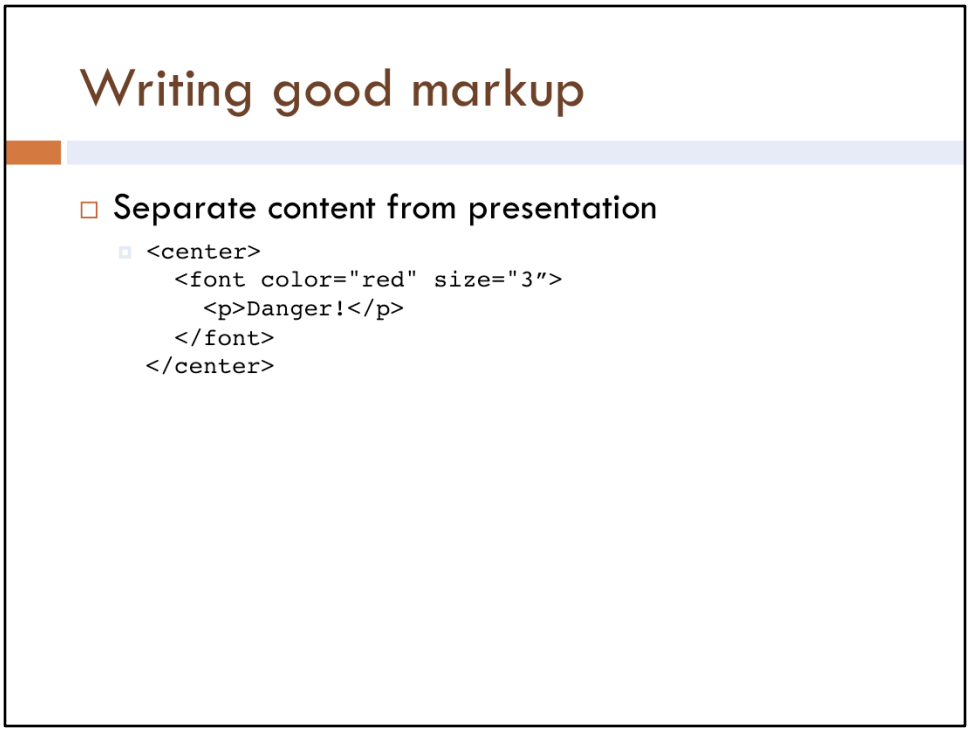
- Maintainability
  - I don't know about you, but I've never been on a project where the requirements don't change, at least a little bit. Every little change has an impact on all levels of the application, and the less time you have to spend reverse engineering your own work or patching unintentional consequences to your changes, the better.
  - Clean, organized markup makes the maintenance job easier; it's easier to find what you need to change, as well as to judge the consequences of the changes you will make.



- Predictability
  - Changing markup can be unpredictable. This has probably happened to you: you make a small change in your markup or styling, and all of a sudden something you didn't touch doesn't look right anymore. This could be caused by anything from coming up against a browser bug, or unknowingly conflicting with a style or behavior you already have in place.
  - Markup that was written and organized by strictly enforcing coding and style guidelines will have—especially in the case of stylesheets—less chance of conflicting with or inheriting unexpected characteristics from other markup, or at the very least be easier to troubleshoot.



- Compatibility
  - Ensuring browser compatibility for your web application is simultaneously the most frustrating and satisfying part of writing markup. Anyone who has spent even a minute wondering why their nice layout is unrecognizable in Internet Explorer will know what I'm talking about.
  - Unsurprisingly, clean, well-organized markup will make this job a lot easier; you'll be able to layer styles and behaviors properly such that your site gracefully degrades, and minimize unpredictable consequences to working code from compatibility-required changes.



- Separate content from presentation wherever possible—this means **everywhere.**
  - Don’t ever put styling in your markup. Say it with me: never put styling in your markup. Use CSS, and put it in a separate file. This will enhance code cleanliness and repeatability.

Writing good markup

- Separate content from presentation

```
<center>
  <font color="red" size="5" >
    <p>Danger!</p>
  </font>
</center>
```

```
<p class="alert">Danger!</p>
```

```
p.alert
{
  color: red;
  font-size: 140%;
  text-align: center;
}
```

This is what we call “semantic” markup; the HTML contains no information about presentation whatsoever.

Writing good markup

- Separate content from presentation

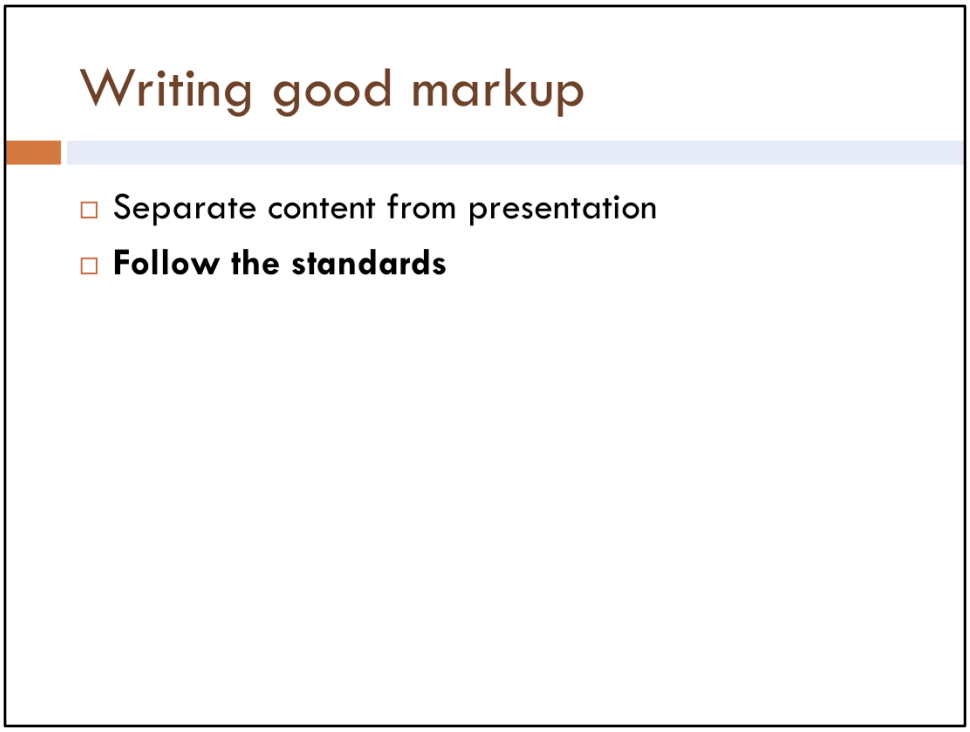
```
<a href="#" onclick="showHide();">
  Show/hide stuff.
</a>

<div id="toggle-me" style="display: none;">
  <p>Stuff.</p>
</div>

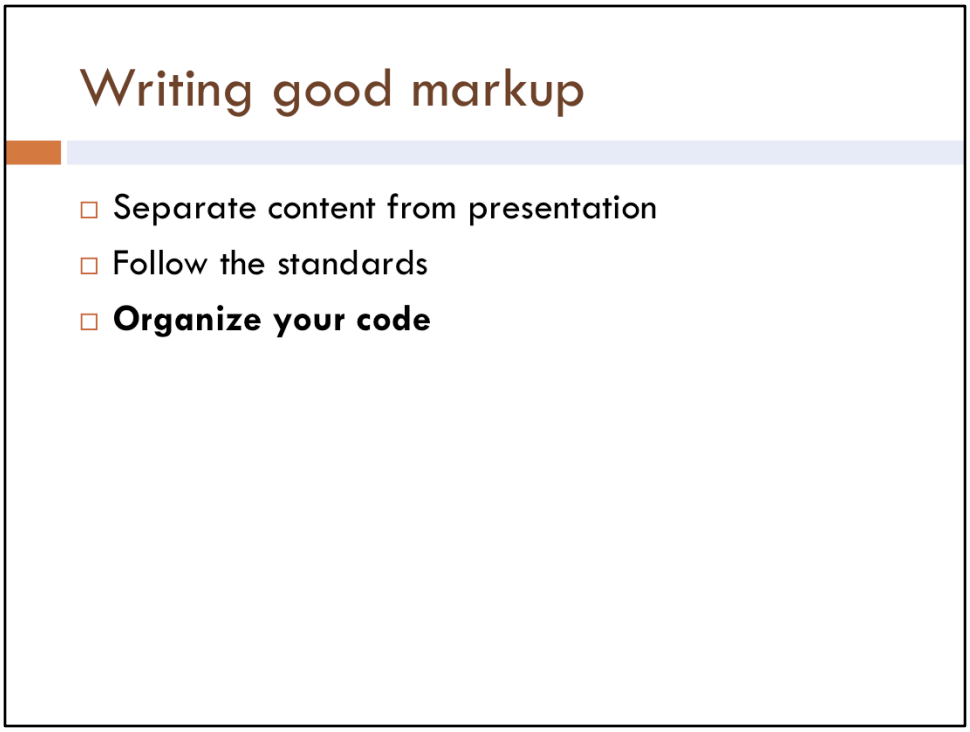
function showHide()
{
    toggle($('toggle-me'));†
}
```

The only exception to this rule is when you're showing/hiding an element dynamically. Then style="display: none;" is permissible.

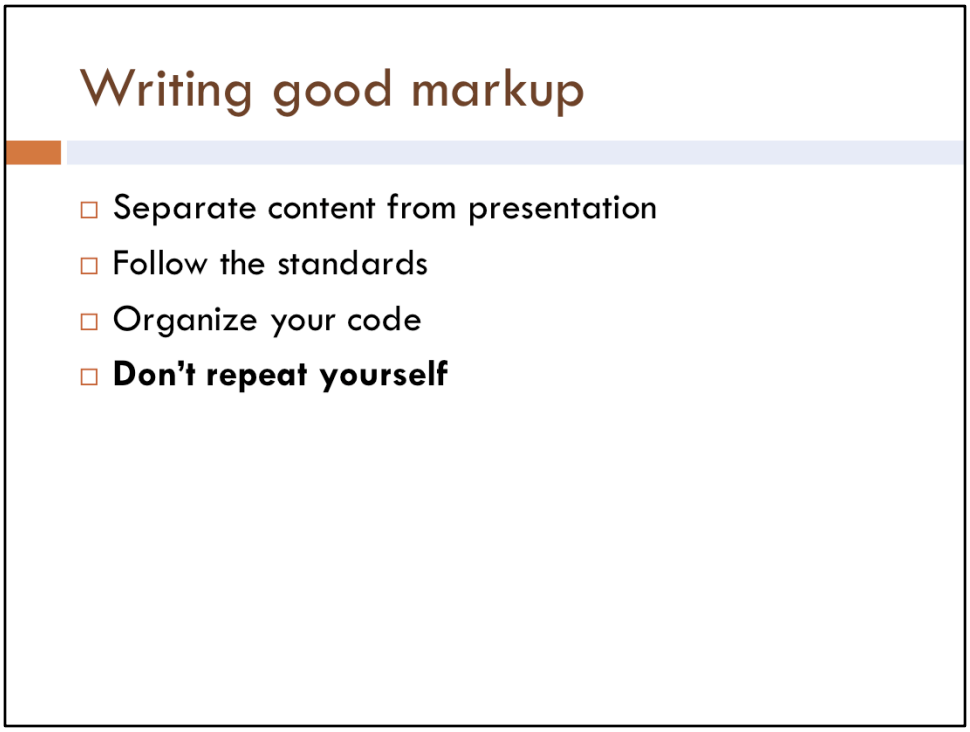
† This code assumes that you've included the Prototype JS library.



- Follow the standards.
  - We'll touch on this in more detail in a bit, but for now, know that the standards are there to make your life easier. Learn them, use them.
  - This course is not going to teach the standards. That's left as an exercise to you. We're more concerned with using the standards *effectively*.



- Organize your code.
  - Coding standards and style guidelines are just as crucial in the markup realm as in technical programming.
  - Organize your code well and you'll get repeatable, predictable, debug-able behavior from your markup.



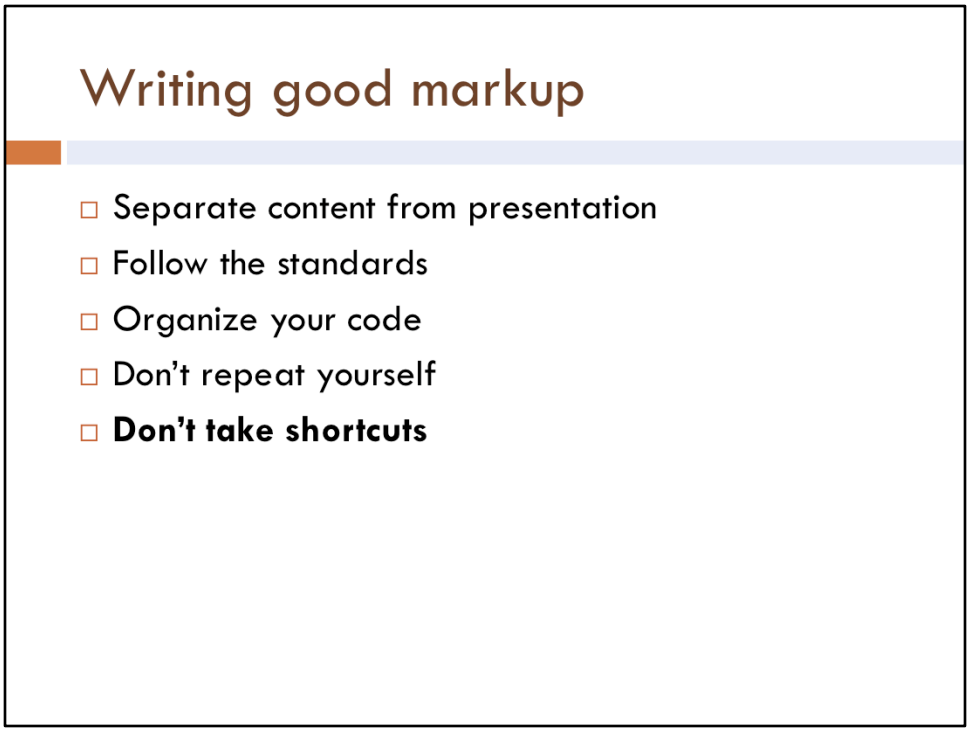
- Don't repeat yourself
  - Again, same as with traditional programming, you want to minimize the amount of repeated code. Make common-sense separations between your styling rules for maximum repeatability and minimum duplicate code.

Writing good markup

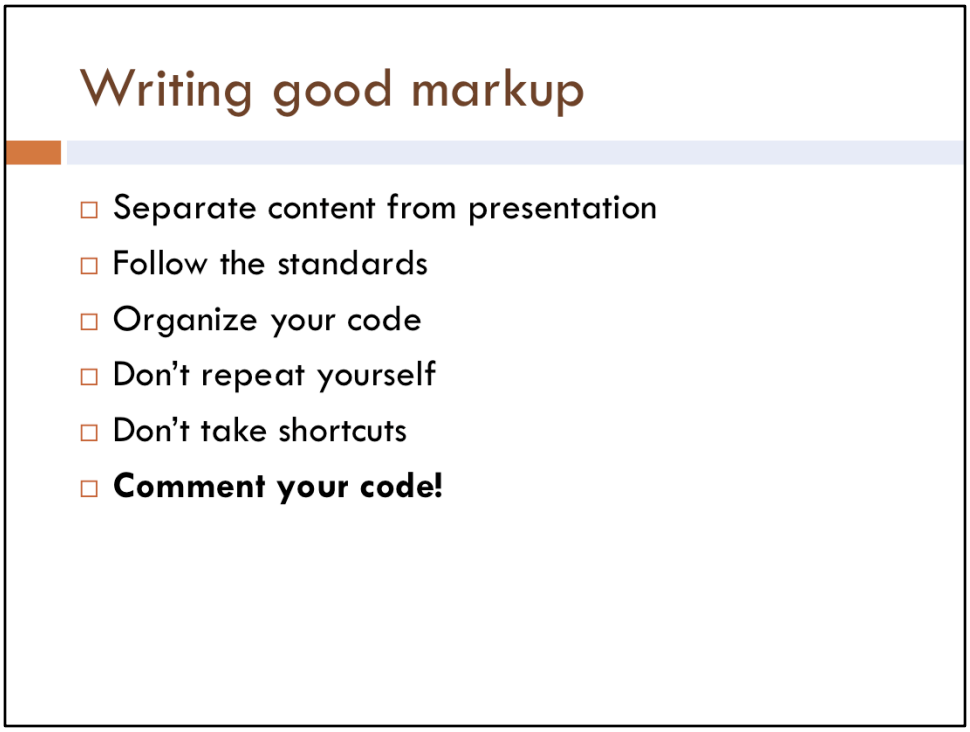
- Separate content from presentation
- Follow the standards
- Organize your code
- **Don't repeat yourself**

```
<p style="background-color: #cccccc;">
    Some content.
</p>
<p style="background-color: #cccccc;">
    Some more content.
</p>
<p style="background-color: #cccccc;">
    Yet more content.
</p>
```

- Don't repeat yourself
  - Again, same as with traditional programming, you want to minimize the amount of repeated code. Make common-sense separations between your styling rules for maximum repeatability and minimum duplicate code.



- Don't take shortcuts
  - I know you're in a hurry, and your code needs to be done yesterday. Doesn't matter. Stop yourself from embedding that bit of styling in the middle of your markup, just because you think don't have time to put it where it belongs. Take the time to do it right the first time, and you'll save yourself time and heartache later.
  - Just like following a process in traditional software engineering.



- Finally, comment your code. Comment your code!
  - Are we seeing a theme here? Treat writing markup like you would traditional programming. Comment your code just as diligently. I'm talking about HTML and CSS as well as JavaScript.
  - In HTML, add comments to closing tags to show what they match up with. Improper nesting can wreck your layout, and is very difficult to pinpoint.
  - Separate CSS into sections by layout and page. Comment each section and add a header to each file.

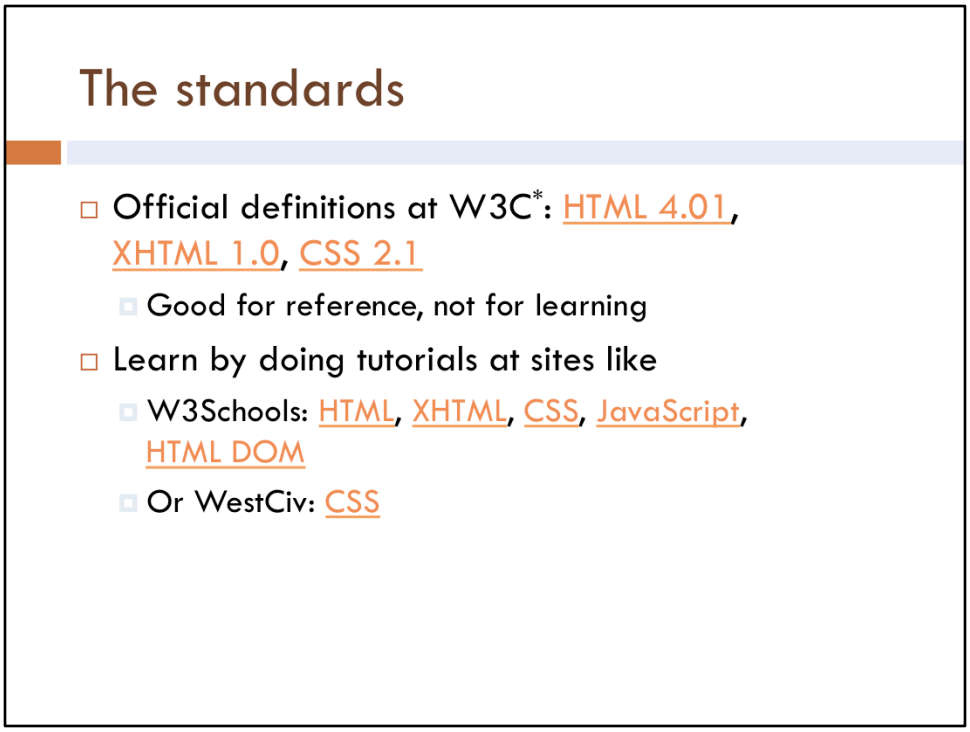
Writing good markup

- Separate content from presentation
- Follow the standards
- Organize your code
- Don't repeat yourself
- Don't take shortcuts
- **Comment your code!**

```
<div class="blah">
…
</div> <!-- /blah -->
```

- Finally, comment your code. Comment your code!
  - Are we seeing a theme here? Treat writing markup like you would traditional programming. Comment your code just as diligently. I'm talking about HTML and CSS as well as JavaScript.
  - In HTML, add comments to closing tags to show what they match up with. Improper nesting can wreck your layout, and is very difficult to pinpoint.
  - Separate CSS into sections by layout and page. Comment each section and add a header to each file.

The standards

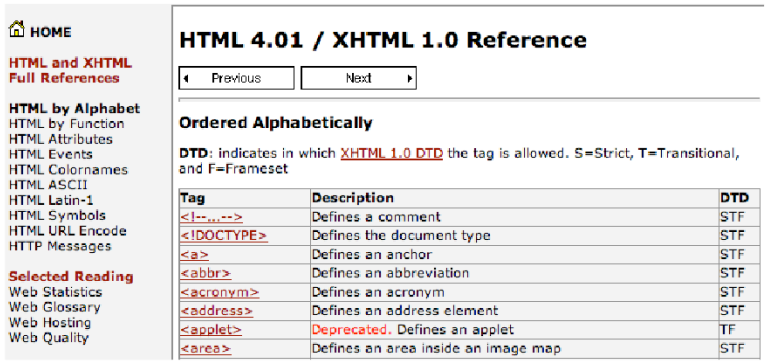


- The official standards specifications can be found at the W3C, but unless you already have a good understanding of how HTML, XHTML and CSS work in practice, they'll be very hard to get through.
    - None of the W3C's specs are the easiest things to read, but when you really need to understand the desired behavior of a content or presentation element, this is where you have to come. So while these specifications are the foundations of the web, they're usually my last line of defense; I find that you have to have a pretty good understanding of how the technology works in practice first before you come here with detailed questions of how it *should work.*
- If the W3C specs are my last line of defense, learning sites like W3Schools are my first. Most people learn best by doing, and the best way in my opinion to learn the web standards is by doing tutorials.

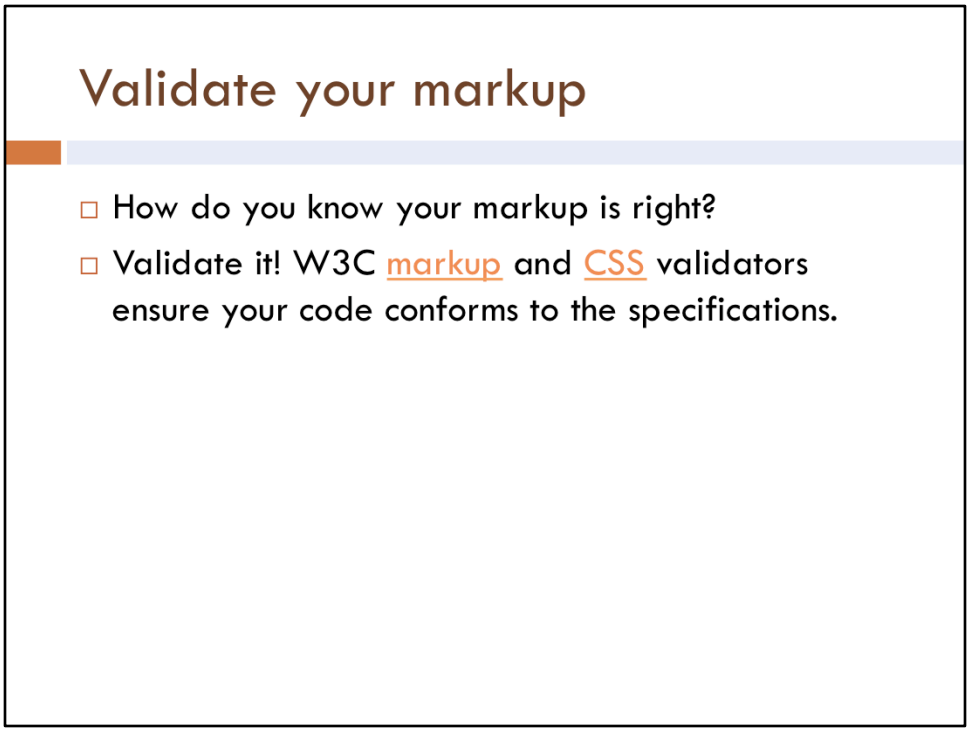
* World Wide Web Consortium

The standards (cont'd)

- W3Schools also provides convenient reference guides



Use the W3Schools reference guides for quick lookups as well.



Now that you've written your markup, how do you check it's right? Validate it; validation is analogous to compilation in traditional programming. The W3C provides validation services for markup and CSS, so you can check that your code conforms to the specifications.

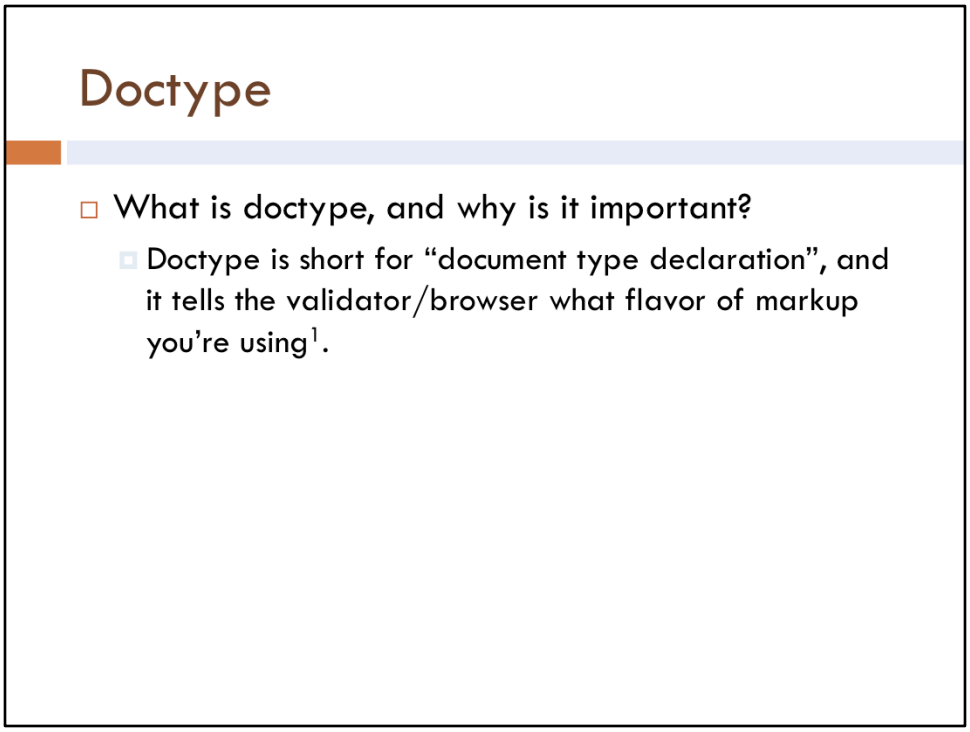
HTML & doctype

HTML vs. XHTML

- HTML 4.01 and XHTML 1.0 are functionally equivalent.
- XHTML follows XML syntax rules and is easier for developers to read.

Doctype

- What is doctype, and why is it important?



- Doctype is used by both the validator and the browser:
  - The validator needs to know what standard you wrote your code to conform to so it picks the right "answer key" to compare to, but modern, standards-compliant browsers also use doctype to know how forgiving they should be when attempting to render your markup. Should they be in standards or quirks mode?
    - A browser in quirks mode—triggered by using an outdated or incomplete doctype, or none at all—assumes that you have written bad, incomplete or invalid markup, and tries to render your markup in a backwards-compatible fashion[1], the end result not being guaranteed to look as you expect.
    - On the other hand, a browser that has been alerted to be in standards mode through use of a recent doctype assumes that your code is written according to the specification supplied in your doctype, and renders it as you would expect.
  - Doctype is essential to ensure that browsers render your code predictably.

[1] Zeldman, Jeffrey. "Fix Your Site With the Right DOCTYPE!" A List Apart. 12 April 2002. <http://www.alistapart.com/articles/doctype/>.

Doctype

- What is doctype, and why is it important?
  - Doctype is short for "document type declaration", and it tells the validator/browser what flavor of markup you're using[1].
  - The doctype must appear at the top of the page
    - ```
      <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
            "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

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

Doctype

- What is doctype, and why is it important?
  - Doctype is short for "document type declaration", and it tells the validator/browser what flavor of markup you're using[1].
  - The doctype must appear at the top of the page
  - Bottom line: if you don't have a doctype, you can't validate your code, and you can't guarantee that it will be rendered correctly. **Don't forget the doctype.**

Doctype (cont'd)

- There are multiple doctypes, each representing a different flavor of markup: HTML 4.01 Strict, XHTML 1.0 Transitional, etc.
  - `<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">`[2]
  - `<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">`

[2] Dubost, Karl. "My Web site is standard! And yours?" W3C Quality Assurance Interest Group.
08 April 2002. <http://www.w3.org/QA/2002/04/Web-Quality>.

Doctype (cont'd)

- There are multiple doctypes, each representing a different flavor of markup: HTML 4.01 Strict, XHTML 1.0 Transitional, etc.
  - `<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">`[2]
  - `<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">`
- Transitional doctypes are for those sites making the shift from older to modern markup[3], and strict doctypes are for markup that strictly adheres to the standards.

- As the names might suggest, transitional doctypes are more forgiving than strict ones, and allow some deprecated tags, like <font> or <u>. Strict doctypes, on the other hand, don't allow any presentational markup at all.
- The presence of a transitional doctype will ensure that the browser renders your code in quirks mode, and is therefore to be avoided if at all possible.

[3] Johansson, Roger. "Transitional vs. Strict Markup." 24 ways to impress your friends. 2005. <http://24ways.org/2005/transitional-vs-strict-markup>.

Doctype (cont'd)

- There are multiple doctypes, each representing a different flavor of markup: HTML 4.01 Strict, XHTML 1.0 Transitional, etc.
  - `<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">`[2]
  - `<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">`
- Transitional doctypes are for those sites making the shift from older to modern markup[3], and strict doctypes are for markup that strictly adheres to the standards.
- The goal: HTML 4.01 Strict/XHTML 1.0 Strict

The ultimate aim is to write standards-compliant code with a strict doctype, for predictable and consistent results.

Content type and encoding

- Doctype isn't enough; you need to specify the content (MIME) type and the file encoding as well.

Content type and encoding let the browser know how to interpret the file you're sending.

Content type and encoding

- Doctype isn't enough; you need to specify the content (MIME) type and the file encoding as well.

Ever seen this kind of thing? That's because the page is not being served with the correct file encoding.

Content type and encoding

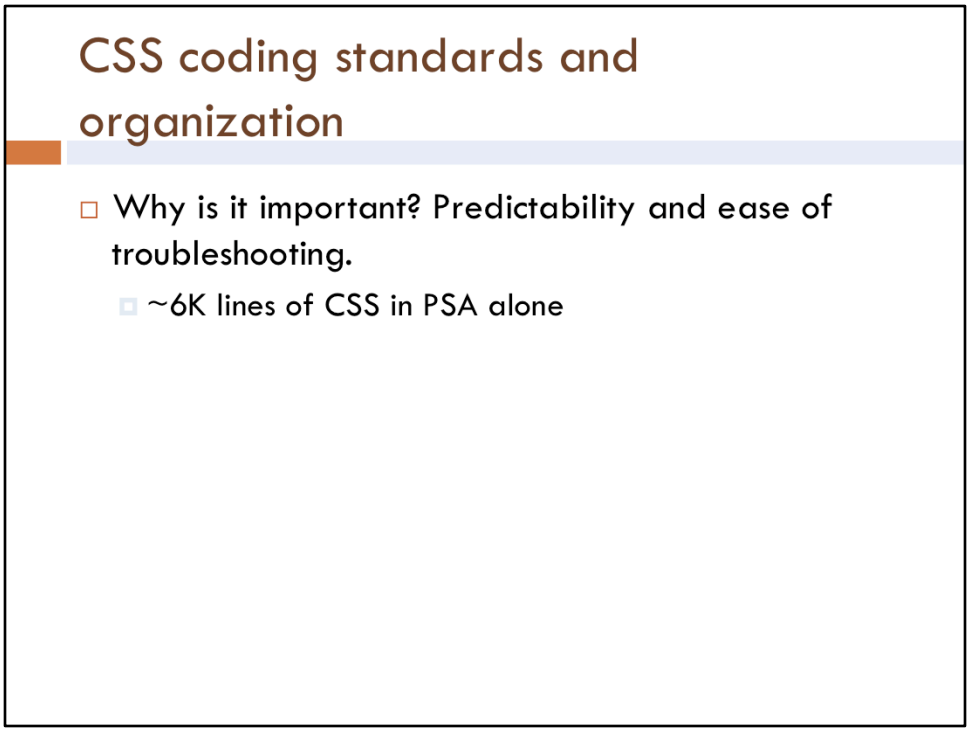
- Doctype isn't enough; you need to specify the content (MIME) type and the file encoding as well.
- They are specified in the meta content type declaration in the header:
  - `<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>`
  - Use `text/html` and `utf-8`[4,5].

[4] Ishikawa, Masayasu. "XHTML Media Types." W3C. 01 August 2002. <http://www.w3.org/TR/xhtml-media-types/>.

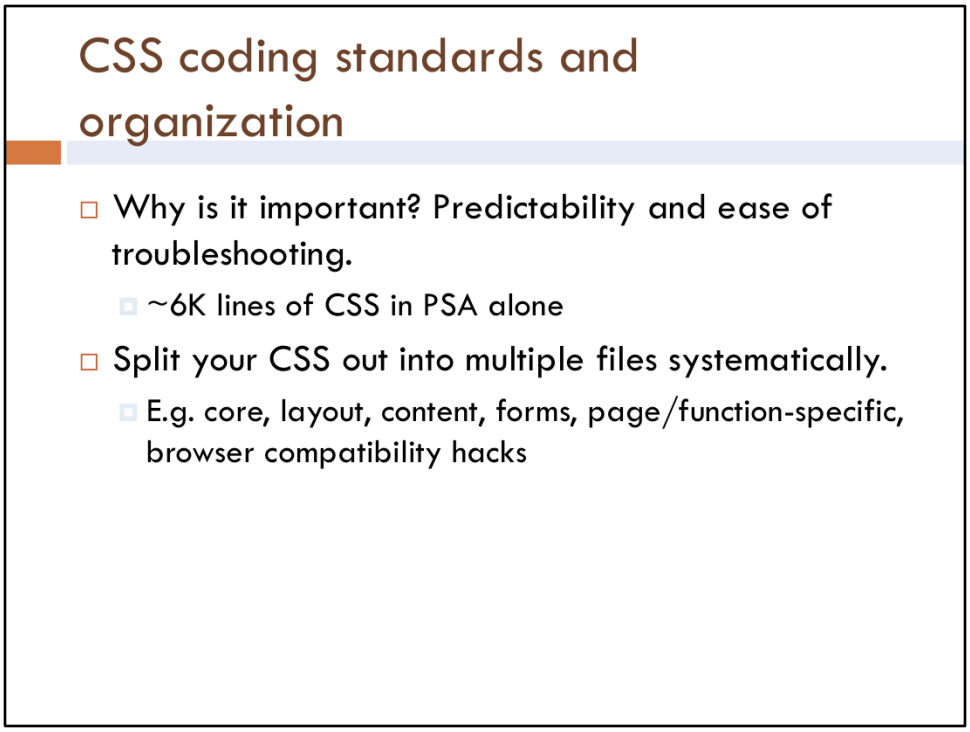
[5] "Frequently Asked Questions About XHTML vs HTML." Sitepoint Forums. 16 June 2006. <http://www.sitepoint.com/forums/showthread.php?t=393445>.

Mini Q&A

CSS coding guidelines



For a project of any reasonable size, your CSS should be strictly compartmentalized and organized. PSA alone has around 6000 lines of CSS code, so you can see that if we weren't exactly sure where our style declarations were at any given time, making changes or debugging problems could become very difficult indeed.



- Your CSS can quickly spiral out of control; keep it well-behaved by splitting it up into several smaller files. Here's how I like to do it:
  - core.css—Put styling for core tags like <h1> and <ul> here.
  - layout.css—Styling for your page layout goes here.
  - content.css—Styling for commonly used content; for example, <p> elements with class "question" should have their styling defined here.
  - forms.css—Styling for form inputs, etc. should be put here.
  - page-specific.css—Styling that is confined to individual pages or functions should be defined here.
  - Finally, the browser compatibility hacks files. This enables us to intelligently include or ignore styling used to address browser shortfalls. Commonly this is just for the various Internet Explorer flavors and can be taken care of by using conditional comments[6].

[6] Shea, Dave. "Stop Hacking, or Be Stopped." Vitamin Features. 23 April 2006. <http://www.thinkvitamin.com/features/css/stop-css-hacking>.

CSS coding standards and organization

- Why is it important? Predictability and ease of troubleshooting.
  - ~6K lines of CSS in PSA alone
- Split your CSS out into multiple files systematically.
  - E.g. core, layout, content, forms, page/function-specific, browser compatibility hacks
- Code to the standards, then hack for compatibility.

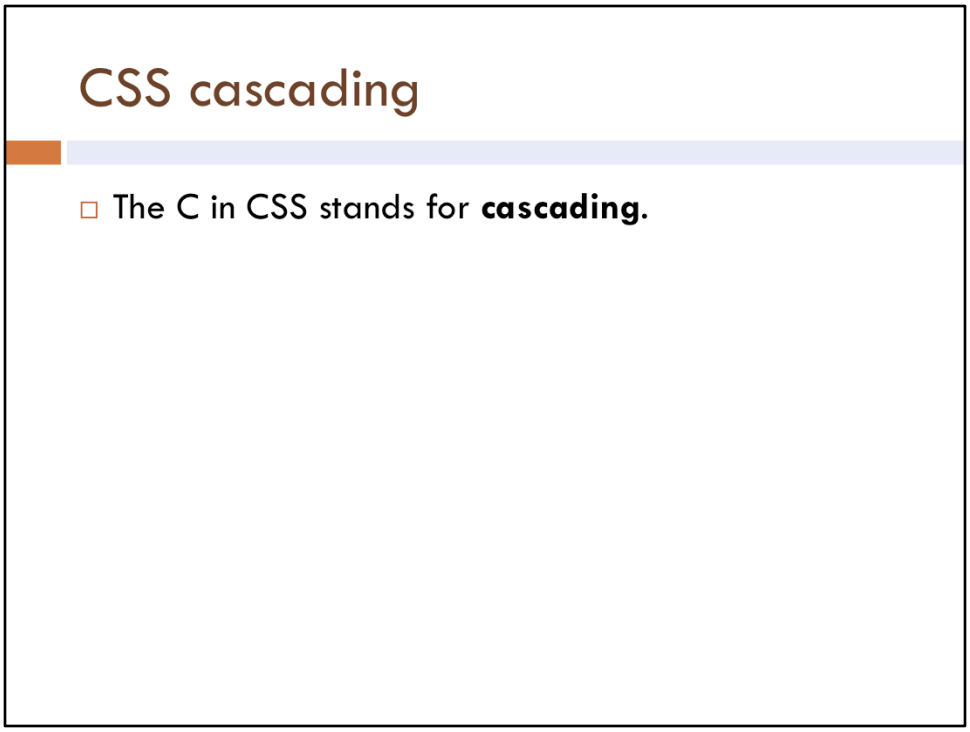
Write code for an ideal environment first, make sure it's working, and *then* worry about compatibility. This allows your code to degrade gracefully across browsers, and your core code base is pure, standards-compliant CSS.

CSS selectors

- In CSS, **selectors** are pattern matching rules that determine which style rules apply to elements in the document tree[7].
  - `p.question` applies to `<p class="question">…</p>`
  - `div#address-form` applies to `<div id="address-form">…</div>`

Selectors are the heart and soul of CSS, one of its subtlest and most difficult concepts, and among the hardest to get right in practice. If an element in the document tree matches a selector, the style rule specified under the selector is applied to the element.

[7] Bos, Bert, Tantek Çelik, Ian Hickson and Håkon Wium Lie. "Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification." W3C. 19 July 2007. <http://www.w3.org/TR/CSS21/>.



The best way to explain cascading is to look at an example.

CSS cascading

- The C in CSS stands for **cascading**.

```
<style type="text/css">
  body
  {
    margin: 0px;
    padding: 10px;
    background-color: #e0e0e0; /*light grey*/
    color: #606060; /*dark grey*/
  }

  p
  {
    margin: 2px;
    padding: 5px;
    background-color: #c0f0c0; /*light green*/
    color: #006600; /*dark green*/
  }

  .greeting
  {
    background-color: #c0c0f0; /*light blue*/
  }
</style>
```

```
<p class="greeting">
  Hello world!
</p>
```

On the left we have a snippet of CSS, and on the right a snippet of HTML. Explain both.

CSS cascading (cont'd)

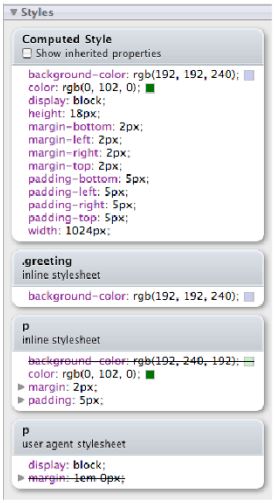
- What is rendered:

CSS cascading (cont'd)

- What is rendered:

Hello world!

- The cascade ➔



From the bottom to the top are the rules that apply to the element. As you can see, multiple style rules apply to this single element; this is the cascade in action.

CSS specificity

- Style rules are ranked by **specificity.**

```
<style type="text/css">
  body
  {
    margin: 0px;
    padding: 10px;
    background-color: #e0e0e0; /*light grey*/
    color: #606060; /*dark grey*/
  }

  p
  {
    margin: 0px;
    padding: 5px;
    background-color: #c0f0c0; /*light green*/
    color: #006600; /*dark green*/
  }

  .greeting
  {
    background-color: #c0c0f0; /*light blue*/
  }

  #intro p
  {
    background-color: #f0c0c0; /*pink*/
  }

  .page-header p
  {
    background-color: #f0f0c0; /*light yellow*/
  }
</style>
```

```
<div class="page-header" id="intro">
  <p class="greeting">
    Hello world!
  </p>
</div>
```

When multiple style rules apply to the same element, how do we know which takes precedence? By measuring specificity.

CSS specificity

- Style rules are ranked by **specificity.**

```
<style type="text/css">
  body
  {
    margin: 0px;
    padding: 10px;
    background-color: #e0e0e0; /*light grey*/
    color: #606060; /*dark grey*/
  }

  p
  {
    margin: 0px;
    padding: 5px;
    background-color: #c0f0c0; /*light green*/
    color: #006600; /*dark green*/
  }

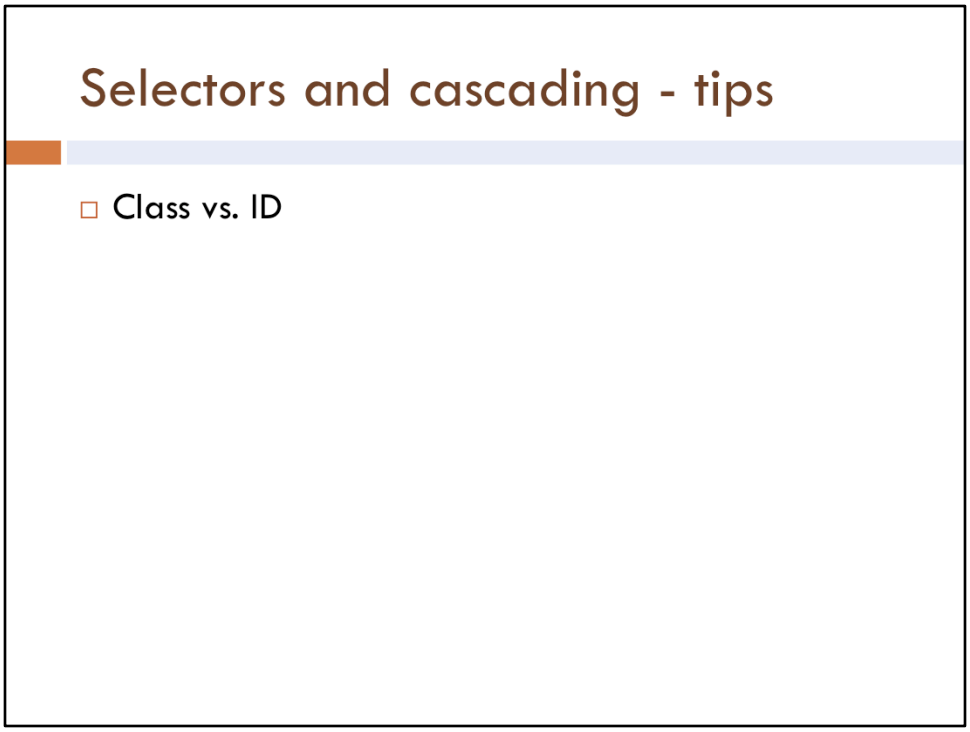
  .greeting
  {
    background-color: #c0c0f0; /*light blue*/
  }

  #intro p
  {
    background-color: #f0c0c0; /*pink*/
  }

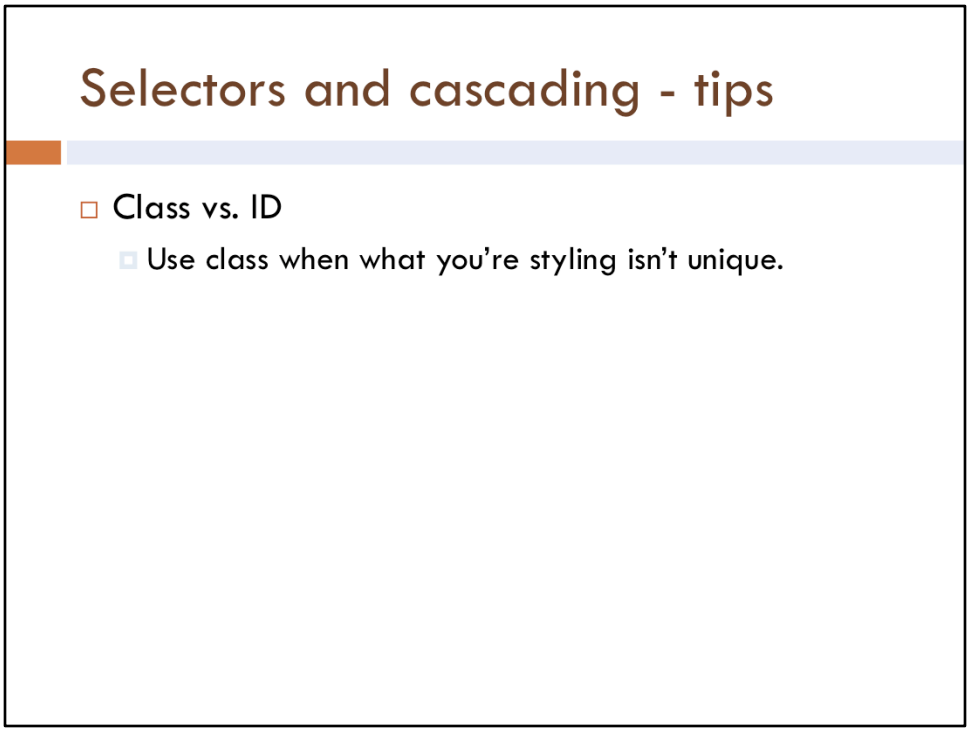
  .page-header p
  {
    background-color: #f0f0c0; /*light yellow*/
  }
</style>
```

```
<div class="page-header" id="intro">
  <p class="greeting">
    Hello world!
  </p>
</div>
```

Hello world!



Class vs. ID: when should you use class selectors, and when should you use ID selectors?



X/HTML requires that IDs are unique on a page. Therefore, only use ID selectors when the element you're styling is unique. If you're going to be using even a little of the same styling elsewhere, consider using a class instead.

Selectors and cascading - tips

- Class vs. ID
  - Use class when what you're styling isn't unique.
  - Give the body element on each page in your application a unique ID.

The body element in each separate page of your application should be given a unique ID. This will help isolate pages when you're doing page/function-specific styling later on.

Selectors and cascading - tips

- Class vs. ID
  - Use class when what you're styling isn't unique.
  - Give the body element on each page in your application a unique ID.
- Complex layouts need a lot of CSS. More CSS = more potential conflicts. How do you minimize them?

Web applications often come with complex layouts. Complex layouts need a lot of CSS. Lots of CSS means there are lots of chances for potential conflicts. How can you as an author minimize them?

Selectors and cascading - tips

- Class vs. ID
  - Use class when what you're styling isn't unique.
  - Give the body element on each page in your application a unique ID.
- Complex layouts need a lot of CSS. More CSS = more potential conflicts. How do you minimize them?
  - Intelligent selector choice

Be smart about selector choice. Recognize and avoid repetition by creating new rules for common styles.

Selectors and cascading - tips

- Class vs. ID
  - Use class when what you're styling isn't unique.
  - Give the body element on each page in your application a unique ID.
- Complex layouts need a lot of CSS. More CSS = more potential conflicts. How do you minimize them?
  - Intelligent selector choice
  - Be minimal; only bring out the big guns when you need to.

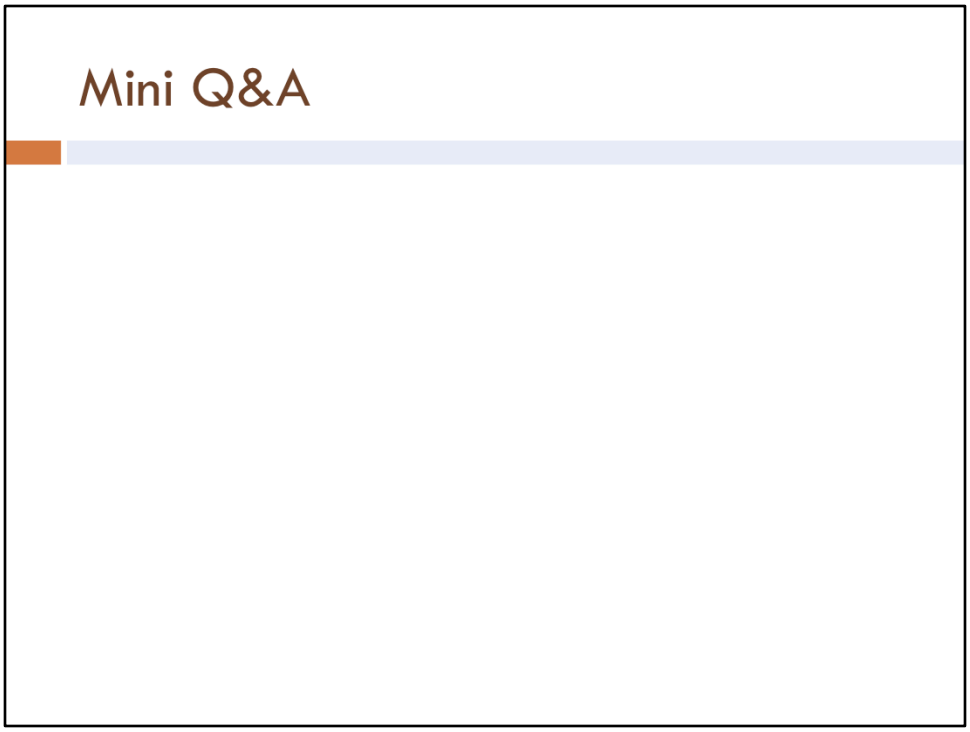
Think minimal; keep the specificity of your rules as low as possible, and don't bring out the big guns (ID selectors) unless you have to. This way you can easily isolate important rules by giving them a high specificity, and you will spend less time locked in a game of one-upmanship with your own style rules.

Interview question vs. FAQ question example.

Selectors and cascading - tips

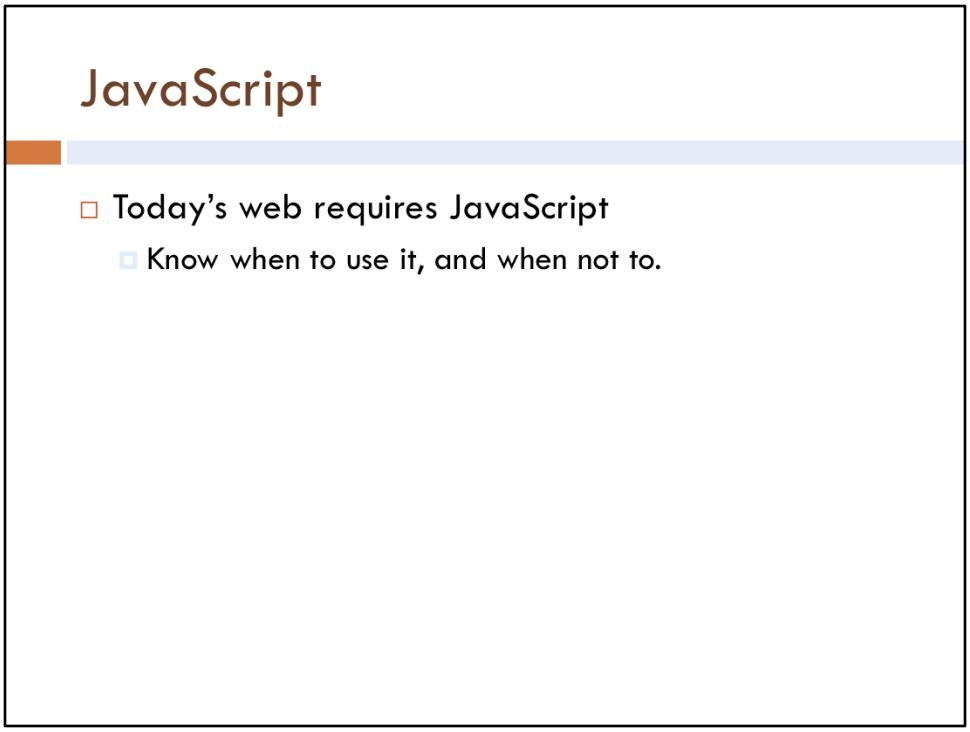
- Class vs. ID
  - Use class when what you're styling isn't unique.
  - Give the body element on each page in your application a unique ID.
- Complex layouts need a lot of CSS. More CSS = more potential conflicts. How do you minimize them?
  - Intelligent selector choice
  - Be minimal; only bring out the big guns when you need to.
  - Conflicts are inevitable. How can you deal with them? Coming up…

Conflicts, however, are inevitable. How do you deal with them? We'll talk about that in the upcoming tools & references section.

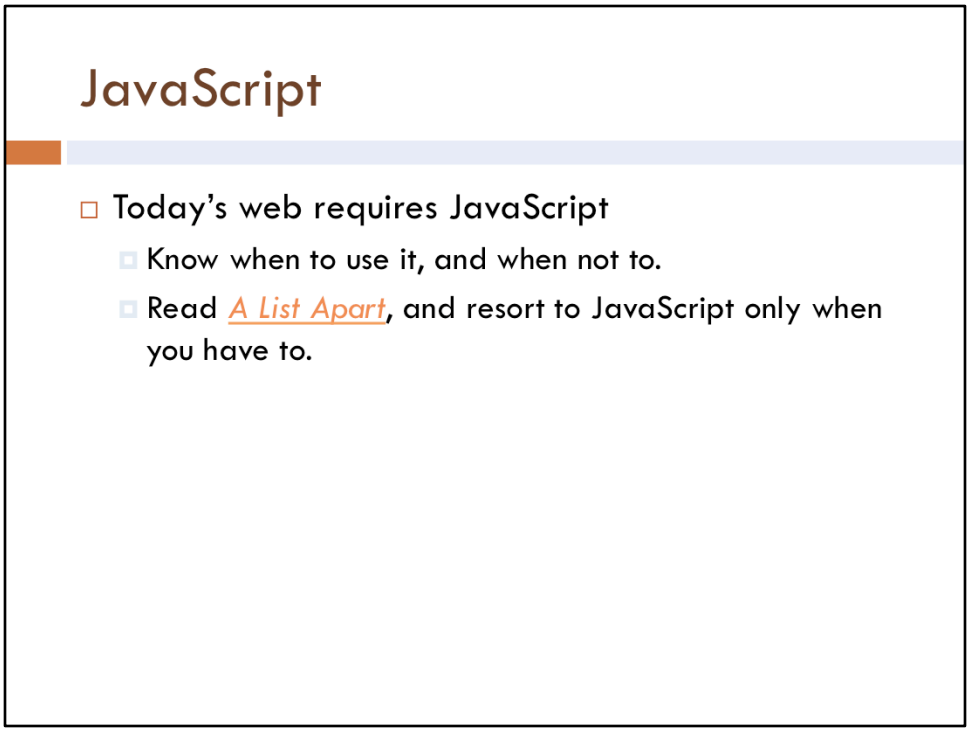


Intermission to follow…

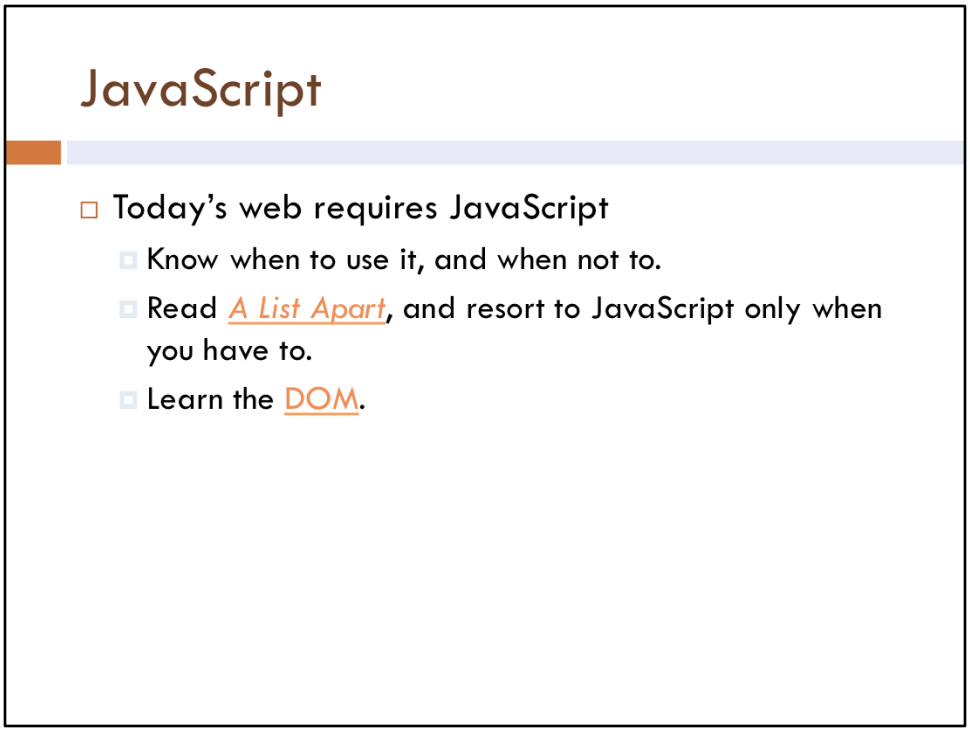
JavaScript



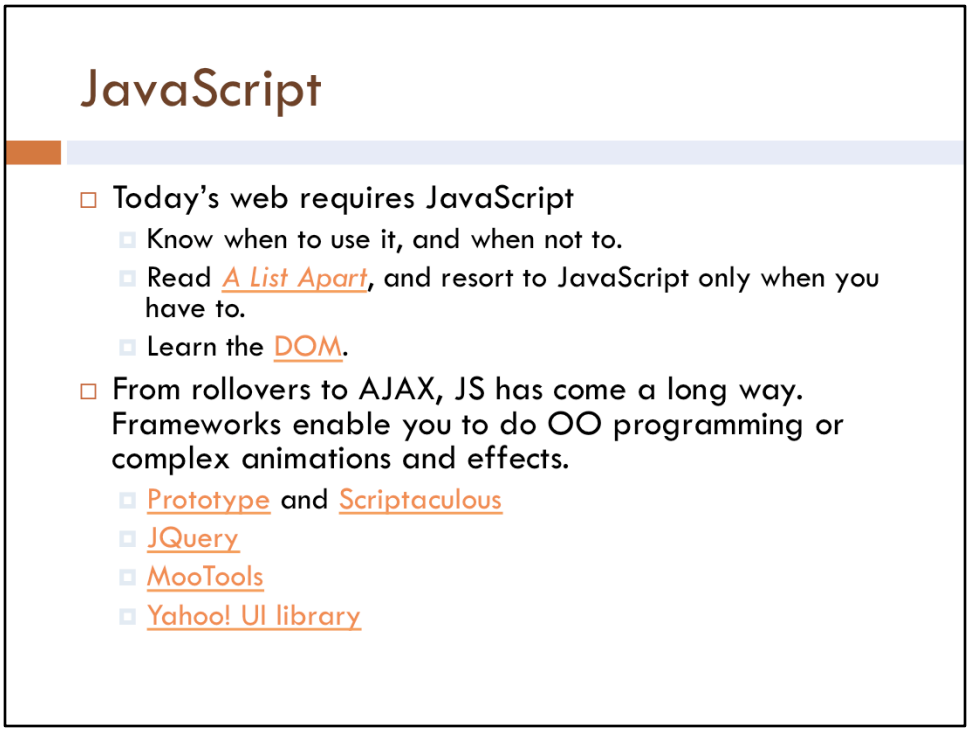
Man does not live by static web pages alone. Today's web requires dynamic, interactive applications, and CSS and HTML will only get you so far. JavaScript is becoming an ever more essential part of any web developer's toolkit, but there's an art to using it.



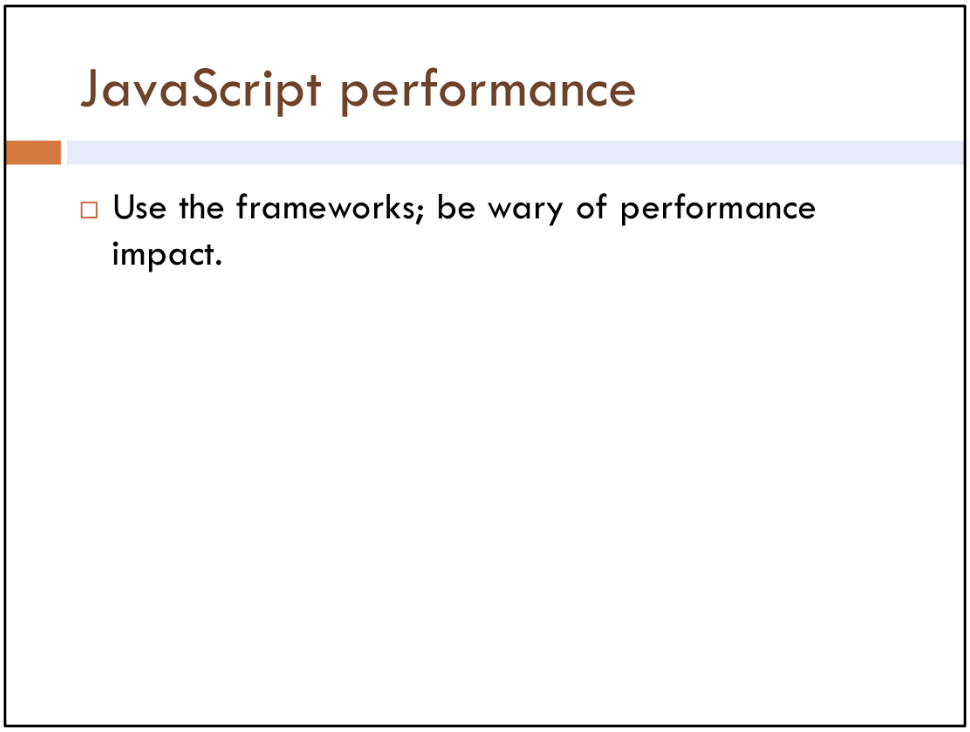
You'd be surprised how much complex layout you can accomplish with CSS alone; just about every common layout can be accomplished without any JavaScript at all. The online publication *A List Apart* has published a number of immensely useful articles on CSS layout that every developer should read before attempting to reinvent the wheel.



Using JavaScript effectively requires a good working knowledge of the HTML Document Object Model (DOM). Learn it.



- JavaScript has evolved quite a bit in recent years, and the advent of JS frameworks has enabled developers to do anything from object-oriented programming to complex animations and effects—all inside a web browser.
    - Prototype and Scriptaculous are perhaps the most well-known JavaScript frameworks/libraries.
    - JQuery focuses on making code easier to write.
    - MooTools is very lightweight.
    - The Yahoo! UI library provides a set of conveniences and UI widgets (like date pickers and auto-complete fields) to make web application programming easier.



The various JS frameworks give you a powerful arsenal of tools, but be aware that just including their code, *even if you don't use it*, can have an adverse impact on your application's performance.

JavaScript performance

- Use the frameworks; be wary of performance impact.
- Most users will hit your application with an empty cache; downloading a lot of files can seriously hurt performance.

No matter how aggressively you cache resources on your server, chances are that a good portion of your users will be coming to your site with an empty cache, necessitating a fresh download of all JavaScript, CSS and image resources. As popular modern browsers like Internet Explorer keep the number of simultaneous HTTP connections to 2 or 4 by default[8], this means many subsequent requests, each with a fixed overhead, regardless of the size of the resource being downloaded. This can seriously hamper site performance, and you haven't even done anything yet.

[8] "HTTP persistent connection." Wikipedia. 30 March 2008. <http://en.wikipedia.org/wiki/HTTP_persistent_connection>.

JavaScript performance

- Use the frameworks; be wary of performance impact.
- Most users will hit your application with an empty cache; downloading a lot of files can seriously hurt performance.
- Moral of the story: strike a good balance between code organization and file minimization.

- How can you work around this problem?
  - Only include what you need.
  - Minimize separate JavaScript file includes especially.

JavaScript performance

- Use the frameworks; be wary of performance impact.
- Most users will hit your application with an empty cache; downloading a lot of files can seriously hurt performance.
- Moral of the story: strike a good balance between code organization and file minimization.
- Browser-specific JS performance discussion coming up…

Now, there are some issues to be aware of regarding JS performance in individual browsers, but we will cover that in our browser compatibility section.

Mini Q&A

Browser compatibility

Browser compatibility

- This is where the fun begins. If you define "fun" as eye surgery. Without anesthesia.
- Ensuring browser compatibility sucks. But you can make the pain bearable.

- Every job has a downside. Browser compatibility is web development's. There you are, with some nice, clean, valid code, and the first time you fire it up in Firefox 2, it doesn't look right. You bring it up in Internet Explorer, and it's unrecognizable. Sadly, success is your only option, so get to work.
- How do you minimize the pain? A few things:

Browser compatibility

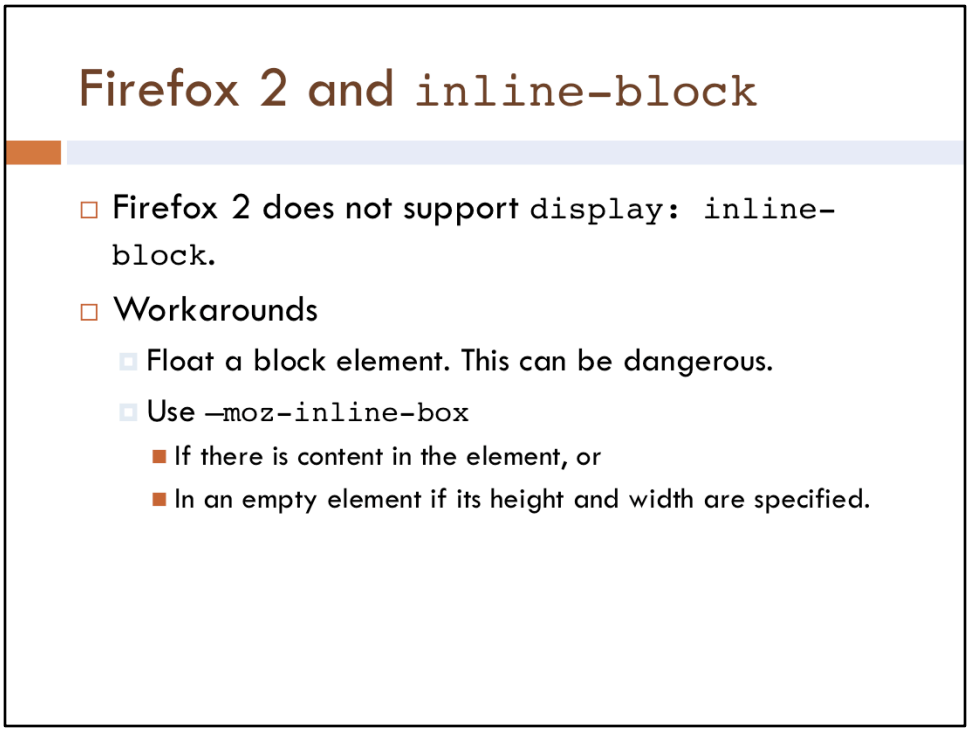
- This is where the fun begins. If you define "fun" as eye surgery. Without anesthesia.
- Ensuring browser compatibility sucks. But you can make the pain bearable.
  - Code to the standards first. Test with Firefox 3, Safari 3, or WebKit. Then worry about (in order): Firefox 2, Internet Explorer 7, Internet Explorer 6.

I said this before: code to the standards, then hack for compatibility. This means initial testing should be done with a browser that fully supports CSS 2, or one that passes the Acid2 test (more on this later). This means Firefox 3, Safari 3 or WebKit. All three of these browsers should render your valid code faithfully, so get your layout working in any of these three first. Then worry about other browsers, in the following order: Firefox 2, IE7, IE6.

Browser compatibility

- This is where the fun begins. If you define "fun" as eye surgery. Without anesthesia.
- Ensuring browser compatibility sucks. But you can make the pain bearable.
  - Code to the standards first. Test with Firefox 3, Safari 3, or WebKit. Then worry about (in order): Firefox 2, Internet Explorer 7, Internet Explorer 6.
  - Be aware of common browser rendering issues.

If you are familiar with common browser rendering issues, you're halfway home. Knowing common issues and their workarounds, and having a good reference library, will be tremendously useful and will keep your hair-pulling to a minimum.



- Sometimes you want to give dimension to your inline elements; this is what the inline-block display mode was intended for. Unfortunately, Firefox 2 does not support it. You can work around the issue in the following ways:
    - Set the element's display to block and float it. However, this path is fraught with danger. IE6 has a whole raft of rendering bugs related to floating elements, so you may be introducing a slew of new problems.
    - Use the proprietary –moz-inline-box display property. This will only work properly if the browser can calculate the dimensions of the element, meaning:
        - The element has content whose height and width is known, or
        - The element is empty but its height and width have been specified.

The Acid2 test

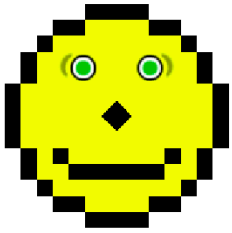
- The Acid2 test is a test page written to help browser vendors ensure proper standards support.

Acid2, properly rendered, will show that the browser supports the standards correctly.

The Acid2 test

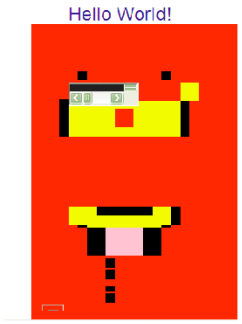
- The Acid2 test is a test page written to help browser vendors ensure proper standards support.
- How it should look:

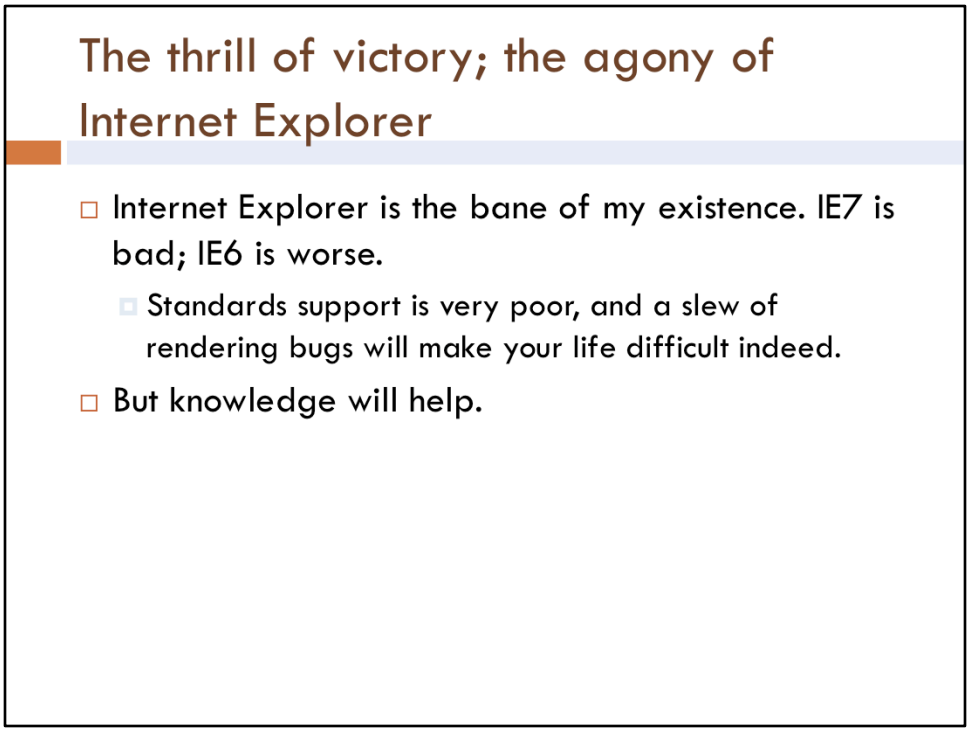
Hello World!



The Acid2 test

- The Acid2 test is a test page written to help browser vendors ensure proper standards support.
- How it looks in IE7:

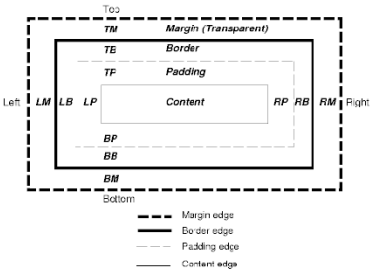


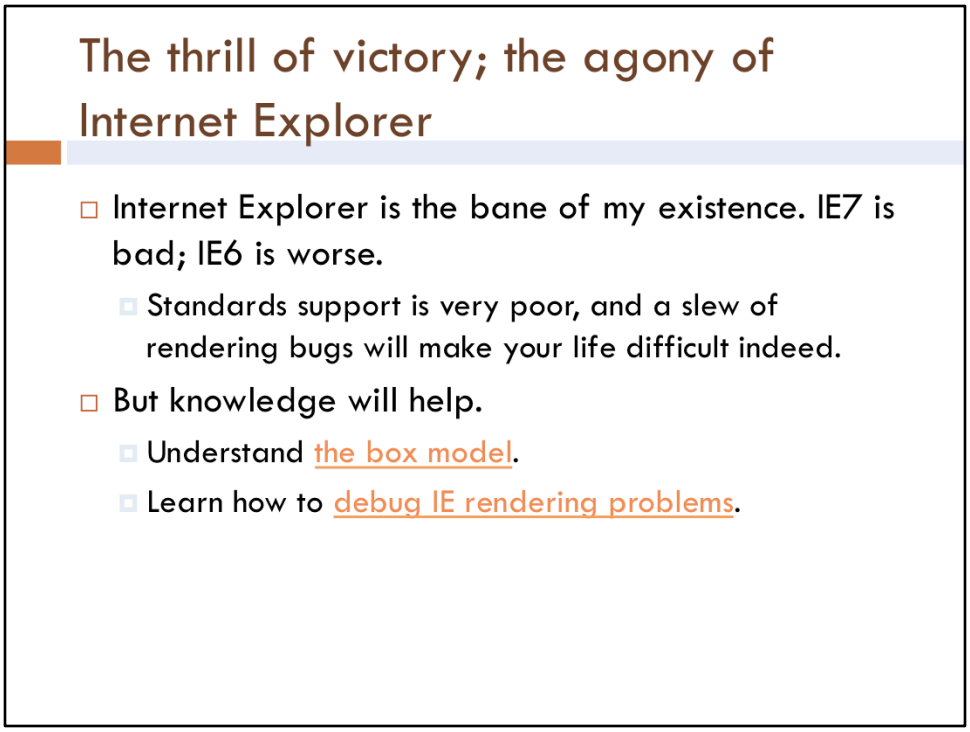


Internet Explorer is still, sadly, the most widely used browser in the world. This means you have to support it, whether you like it or not. I'm guessing not. You'll find that once you have your website working in Safari and Firefox, your job is only partially done. If you have to support IE6, you're not even halfway there. Working with either flavor of IE is a test of patience and fortitude, but it won't kill you. If you learn the following things, you will be well armed for your battle:

The thrill of victory; the agony of Internet Explorer

- Internet Explorer is the bane of my existence. IE7 is bad; IE6 is worse.
  - Standards support is very poor, and a slew of rendering bugs will make your life difficult indeed.
- But knowledge will help.
  - Understand the box model.

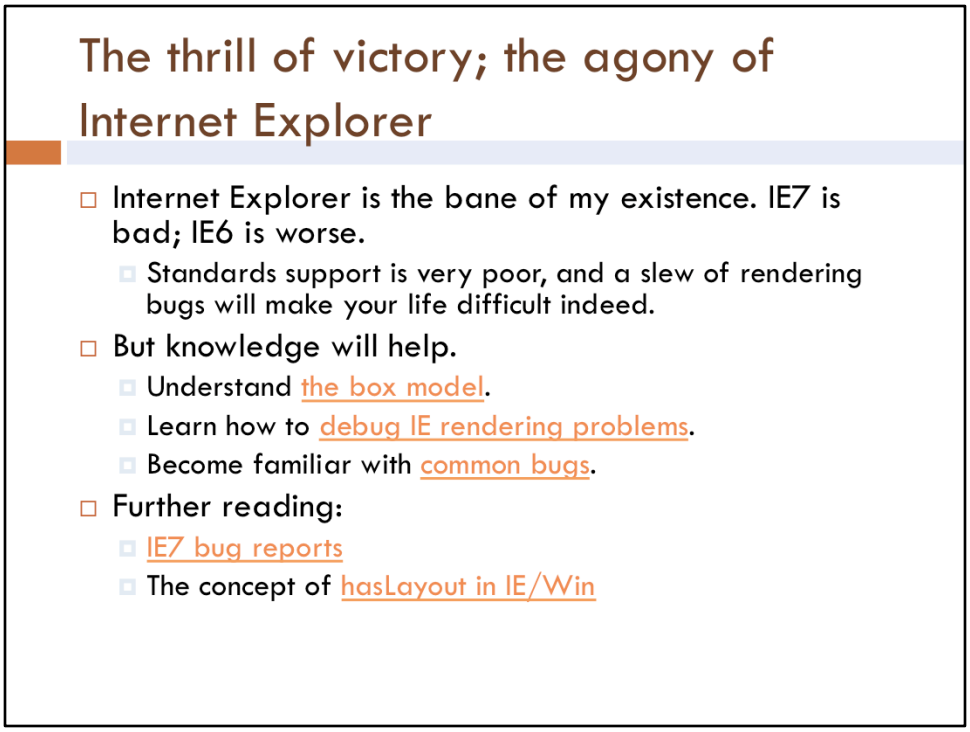




The best way to attack IE display bugs, as described in this article by John Gallant and Holly Bergevin.

The thrill of victory; the agony of Internet Explorer

- Internet Explorer is the bane of my existence. IE7 is bad; IE6 is worse.
  - Standards support is very poor, and a slew of rendering bugs will make your life difficult indeed.
- But knowledge will help.
  - Understand the box model.
  - Learn how to debug IE rendering problems.
  - Become familiar with common bugs.



Also read about reported bugs in IE7, as well as the concept of having layout in Internet Explorer.

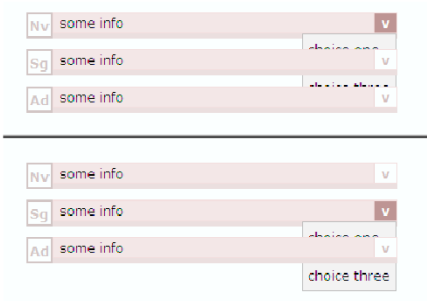
IE and the z-index bug

- The z-index bug has been around since IE 4 or 5, and has *still* not been fixed.

How many people here know what the z-index property is for? Since positioned elements can fall out of the normal page layout, it's useful to be able to assign the stacking order of positioned elements in relation to the rest of the page; hence the z-index property. Elements with a higher z-index are rendered "on top of" elements with a lower or automatic z-index value.

IE and the z-index bug

- The z-index bug has been around since IE 4 or 5, and has *still* not been fixed.
- IE creates a new stacking context for elements with *any* z-index value, including auto.



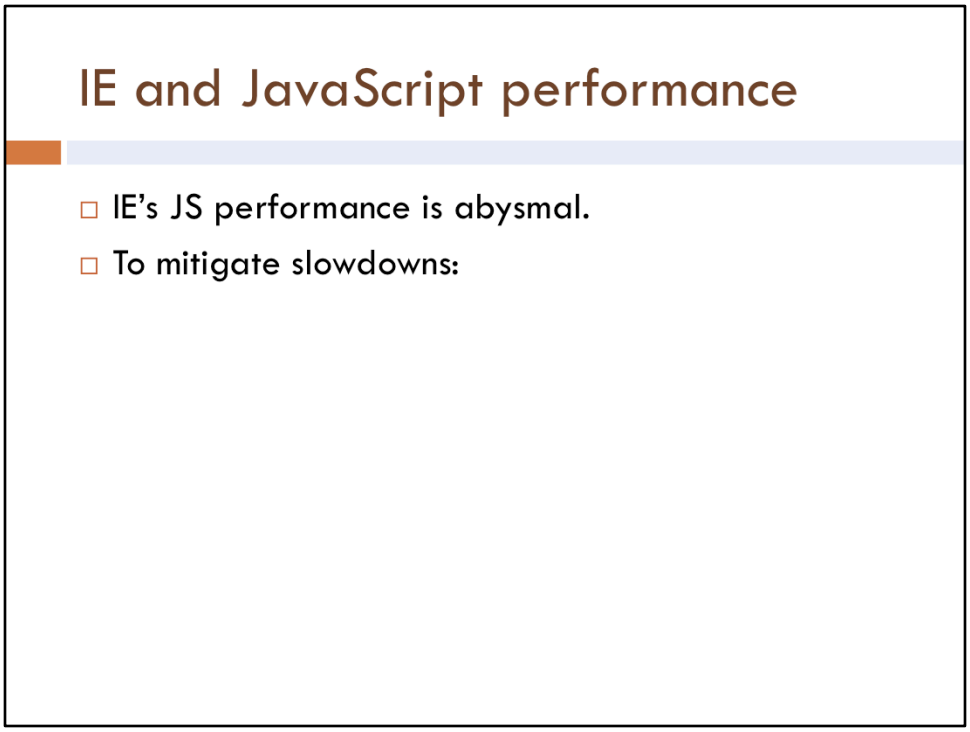
- "The CSS 2.1 spec says that a positioned element with any integer z-index value (i.e. not auto) should create its own zero-based stacking context, and use the integer value specified to decide its place in its parent stacking context. In other words, if the positioned element has a z-index of auto, its stacking context is inherited from its parent. Internet Explorer, however, creates a new stacking context for elements with *any* z-index value, including auto, which wreaks all kinds of havoc and generally causes mayhem in your previously neat and orderly layouts."[9]
- This screenshot is from an early UI design for PSA's dynamic assembly feature. Each of the clip blocks here has a flyout menu attached which has an absolute position and an integer z-index value, so that the flyout appears above all subsequent elements. However, since each clip block is relatively positioned, IE7 renders each subsequent block on top of everything that has come before.

[9] Avasthi, Richa. "IE7 lessons learned: the z-index bug." The Emotional Pumpkin. 11 January 2008. <http://richa.avasthi.name/blogs/tepumpkin/2008/01/11/ie7-lessons-learned/>.

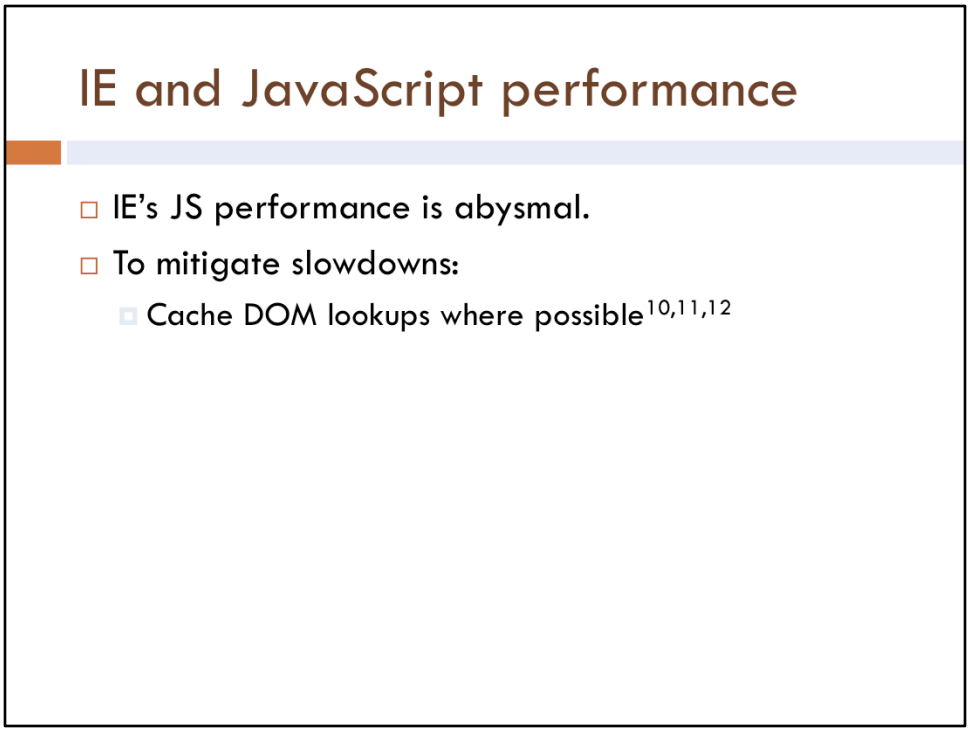
IE and the z-index bug

- The z-index bug has been around since IE 4 or 5, and has *still* not been fixed.
- IE creates a new stacking context for elements with *any* z-index value, including auto.
- Workaround: manually assign unique z-index values to every positioned element on your page.

The way to work around this problem is to manually assign unique z-index values to every positioned element on your page. It's a brute force method, but is the only guaranteed way of ensuring proper stacking of overlapping positioned elements in Internet Explorer.



IE's JS performance is abysmal. In practice, I've observed that running a script in IE takes anywhere between two and *ten* times as long as it would in Firefox or Safari. It'll never be as fast as the other two major browsers, but you can optimize your code so the performance isn't unbearably slow.

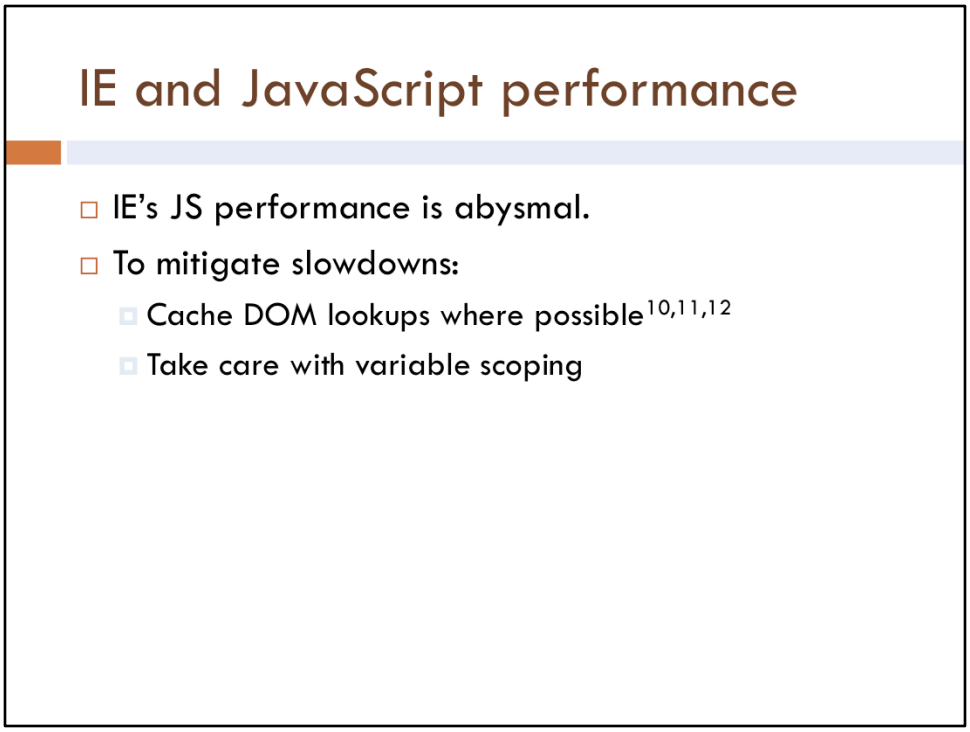


DOM lookups are very expensive in IE; cache them wherever you can.

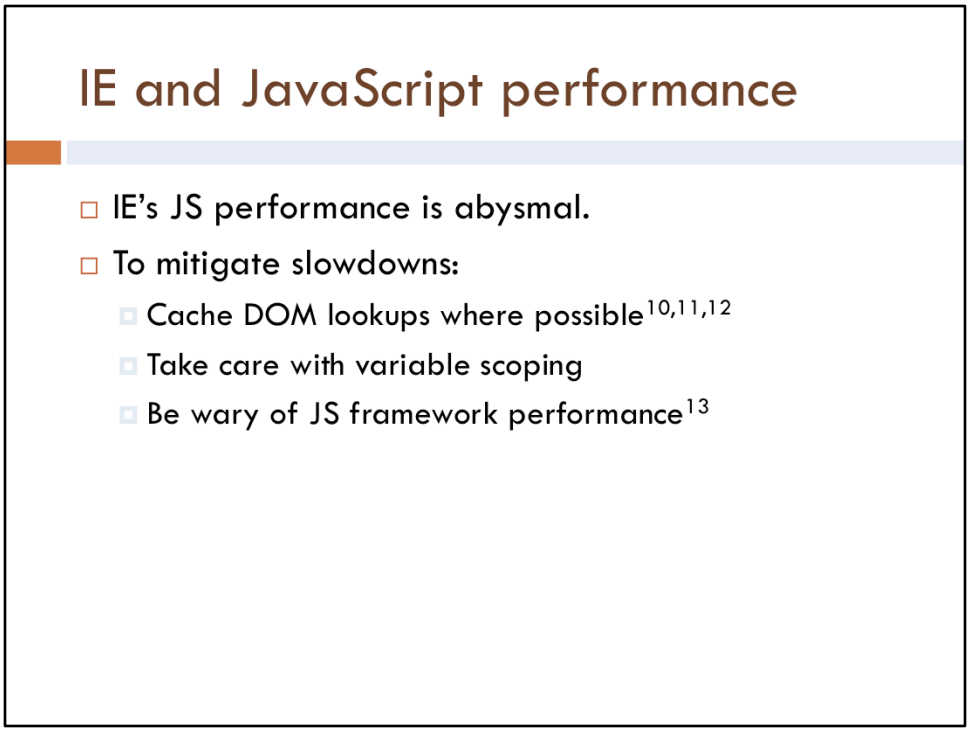
[10] Gurevich, Peter. "IE + JavaScript Performance Recommendations - Part 1." IEBlog. 28 August 2006. <http://blogs.msdn.com/ie/archive/2006/08/28/728654.aspx>.

[11] Gurevich, Peter. "IE+JavaScript Performance Recommendations Part 2: JavaScript Code Inefficiencies." IEBlog. 16 November 2006. <http://blogs.msdn.com/ie/archive/2006/11/16/ie-javascript-performance-recommendations-part-2-javascript-code-inefficiencies.aspx>.

[12] Gurevich, Peter. "IE+JScript Performance Recommendations Part 3: JavaScript Code Inefficiencies." IEBlog. 04 January 2007. <http://blogs.msdn.com/ie/archive/2007/01/04/ie-jscript-performance-recommendations-part-3-javascript-code-inefficiencies.aspx>.



- Be careful with variable scoping:
  - Always use the `var` keyword with local variables. IE will search for global variables if the var keyword isn't used.
  - Make lookups local wherever you can: cache function pointers and references into the DOM tree.



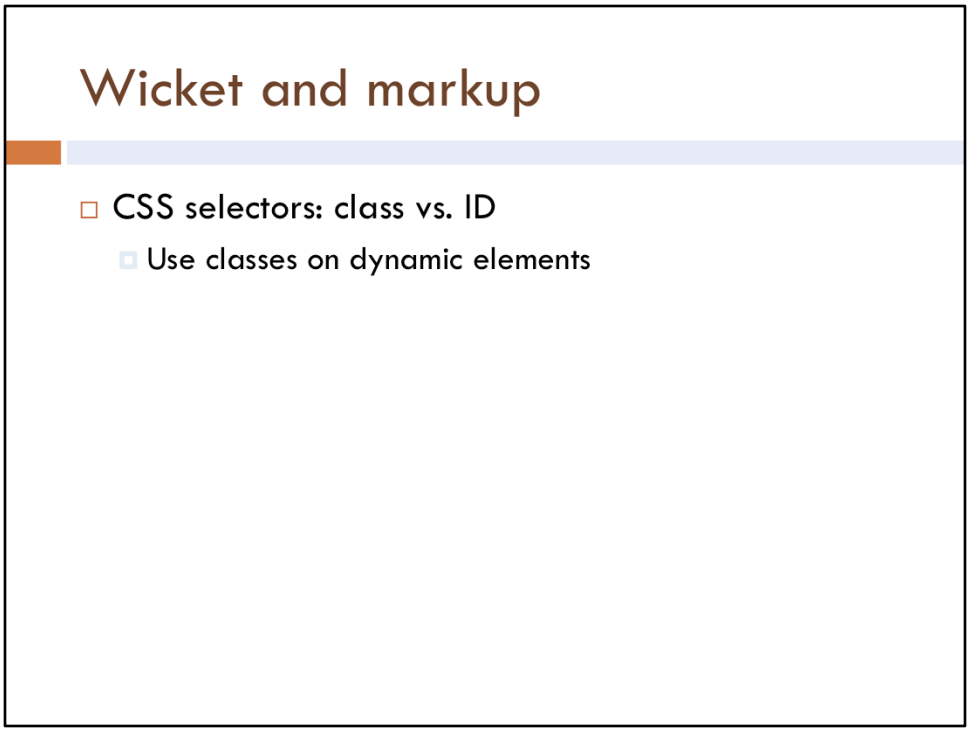
JS framework performance is not always optimal. For example, the `getElementsByClassName` function provided in the Prototype library is horrendously slow. Use an optimized version[14].

[13] Avasthi, Richa. "Talking shop: IE and JavaScript performance." The Emotional Pumpkin. 07 March 2007. <http://richa.avasthi.name/blogs/tepumpkin/2007/03/07/talking-shop-ie-and-javascript-performance/>.

[14] Nyman, Robert. "The Ultimate GetElementsByClassName, Anno 2008." Robert's Talk. 27 May 2008. <http://www.robertnyman.com/2008/05/27/the-ultimate-getelementsbyclassname-anno-2008/>

Mini Q&A

Wicket and markup



What happens when you need to style an element generated by Wicket? What's the best kind of selector you can use? Since many dynamic elements—especially Ajax controls—require a unique markup ID, it is best in the general case to use classes instead of IDs as selectors on dynamic/Wicket-generated elements.

Wicket and markup

- CSS selectors: class vs. ID
  - Use classes on dynamic elements
- How can I hide markup wrapping an element?
  - `setRenderBodyOnly(true);`

Sometimes you don't want to render the tag for your Wicket component; just what it contains. Calling the setRenderBodyOnly() method with true as the argument will render the body of the component only, not its tag.

Wicket and markup

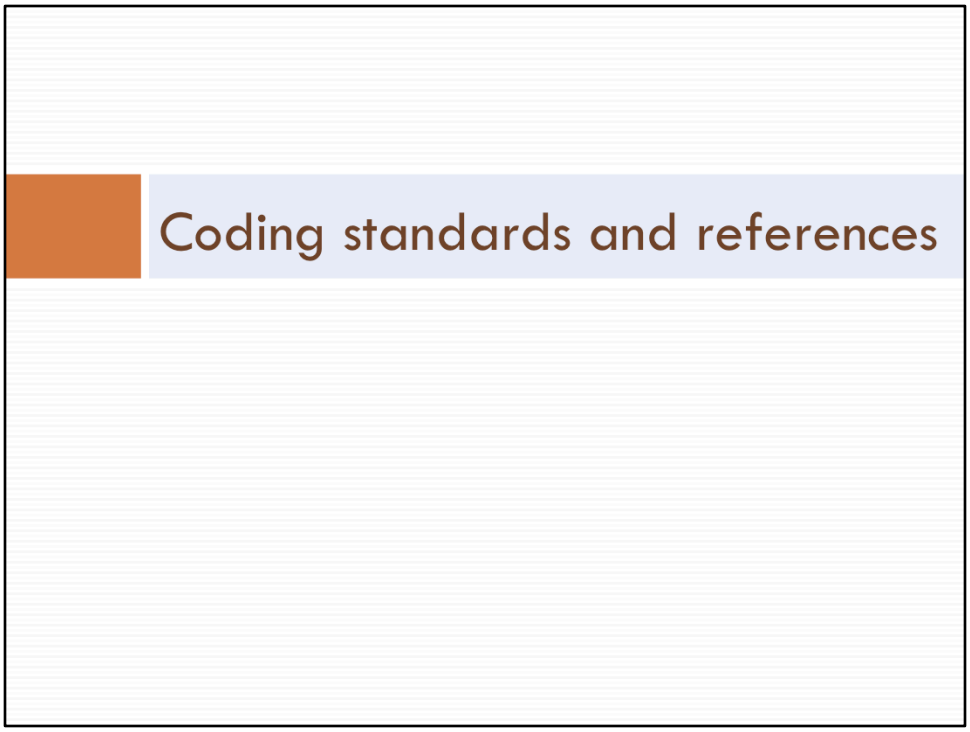
- CSS selectors: class vs. ID
  - Use classes on dynamic elements
- How can I hide markup wrapping an element?
  - `setRenderBodyOnly(true);`
- How can I hide the Wicket markup?
  - `getMarkupSettings().setStripWicketTags(true);`

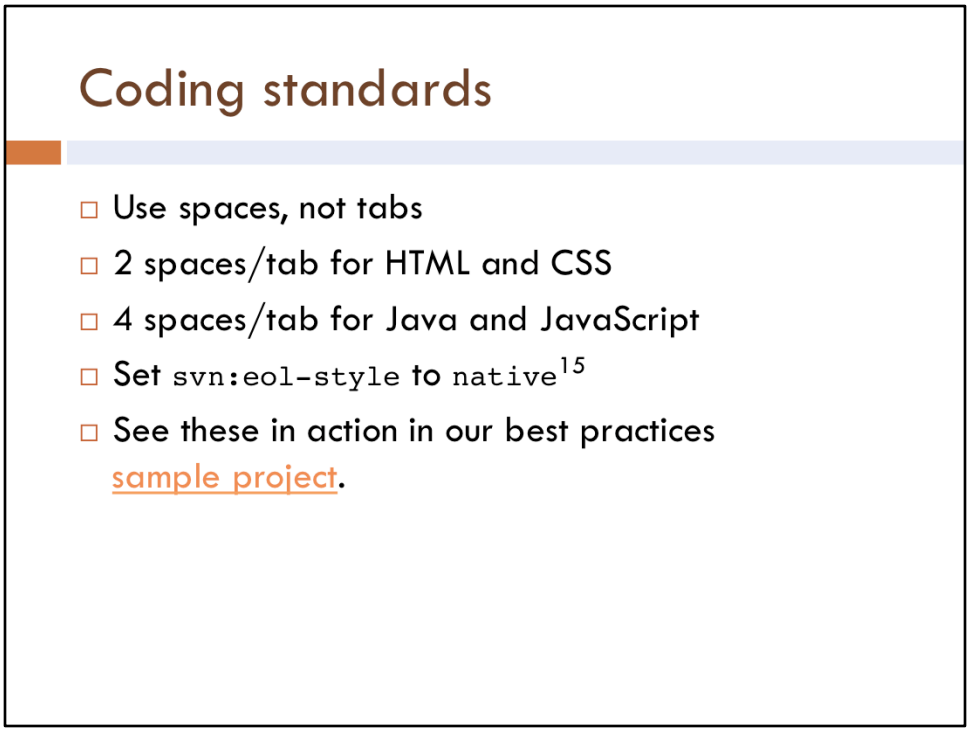
- The Wicket markup (e.g. wicket:id) can cause rendering problems, and will cause your markup to fail validation.
  - Call getMarkupSettings().setStripWicketTags(true) in the init method of your Application class to hide it.

Wicket and markup

- CSS selectors: class vs. ID
  - Use classes on dynamic elements
- How can I hide markup wrapping an element?
  - `setRenderBodyOnly(true);`
- How can I hide the Wicket markup?
  - `getMarkupSettings().setStripWicketTags(true);`
- The CSS/JS caching problem
  - One solution: append version numbers to the files dynamically
  - In Wicket: extend `MarkupParserFactory`

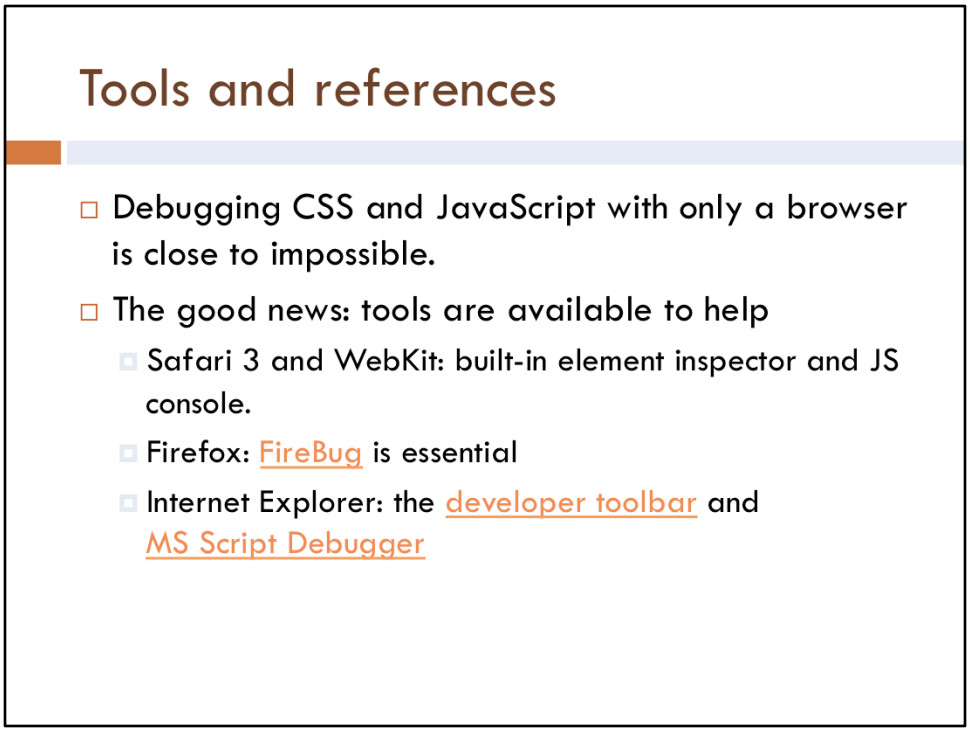
- You want CSS/JS caching to work intelligently and seamlessly; your users should always have the latest version of the stylesheets without having to download them each time.
  - One way to help minimize trouble—the solution we use in PSA—is to append a version number to the linked resource dynamically, so that the application is always requesting the current version of the file. If the version number changes, a download is forced.
  - To do this in Wicket, extend the MarkupParserFactory class.


Coding standards and references

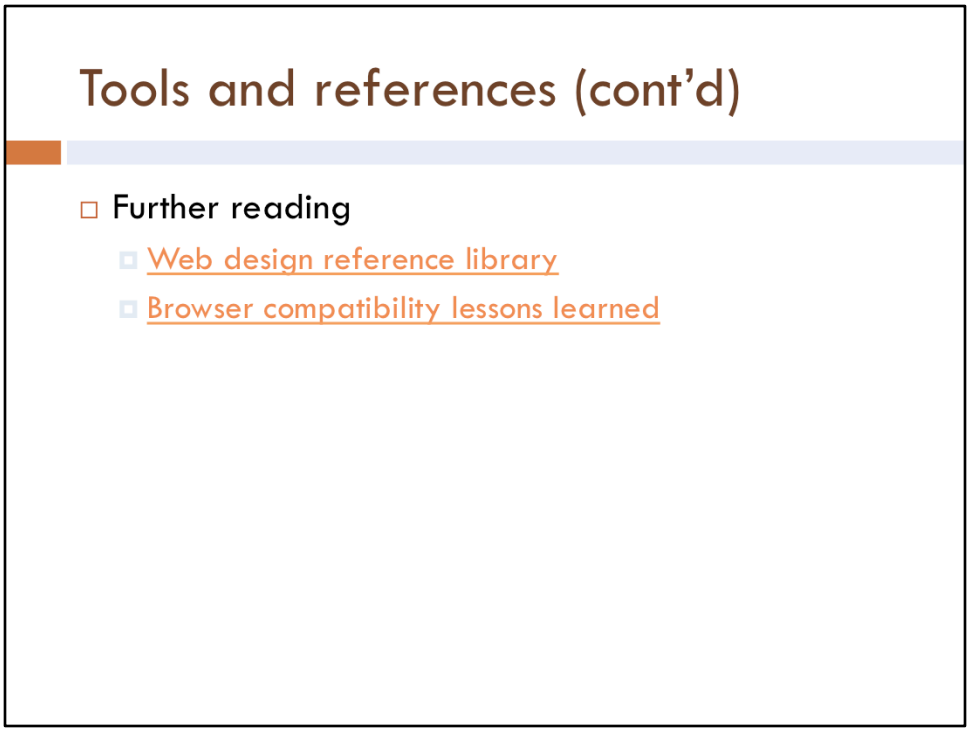


- Here are some coding standards that we recommend:
  - Use spaces instead of tabs; this ensures easier cross-platform portability
  - Use 2 spaces/tab for HTML and CSS
  - Use 4 spaces/tab for Java and JavaScript
  - Your developers don't always work on the same platform. Minimize pain by telling Subversion to automatically convert line endings on check in and check out.
  - Refer to our sample project to see these in action.

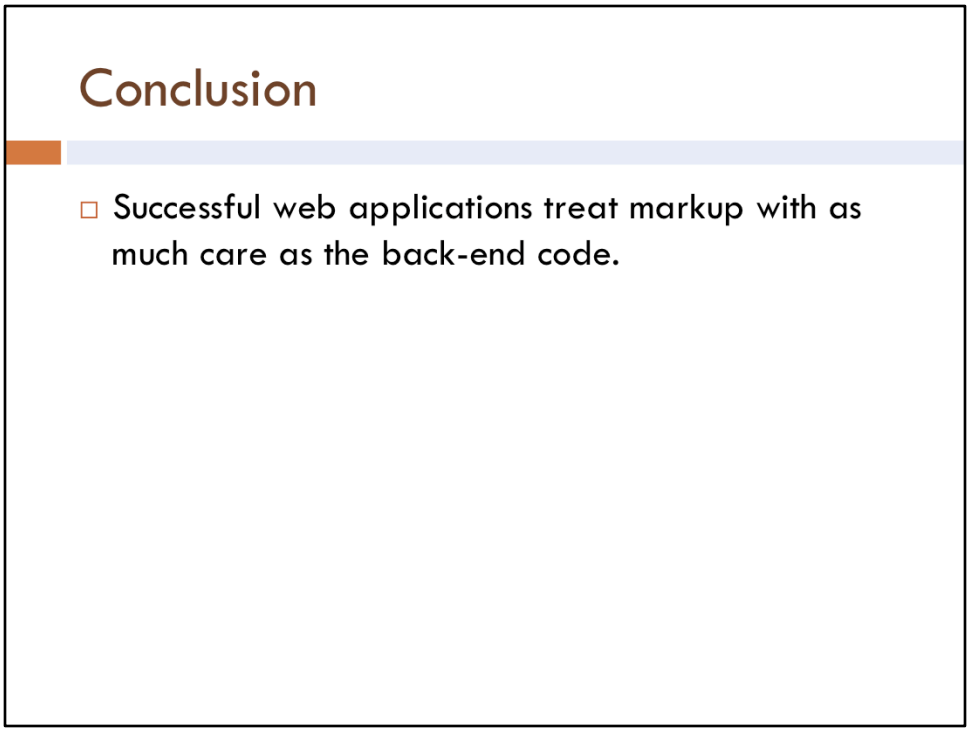
[15] Collins-Sussman, Ben. "Re: Setting svn:eol-style." E-mail to P. Rusk. 03 December 2003.
SVN Users. <http://svn.haxx.se/users/archive-2003-12/0063.shtml>.



- Debugging CSS and JavaScript—complex CSS and JavaScript—is pretty close to impossible with just a browser. You need, at the minimum, some way of knowing the calculated style of your elements, the way individual style rules cascade, and a way to pinpoint JavaScript errors.
- Fear not, though; there are a variety of useful tools for the three major browsers:
    - Safari 3 and WebKit have a very useful built-in element inspector and JavaScript console.
    - For Firefox, the Firebug add-on is absolutely essential. It helps to debug both JavaScript and CSS.
    - Debugging in Internet Explorer, not surprisingly, is actually impossible without the help of a couple of utilities.
        - The first is the IE developer toolbar, to check calculated styles and generated markup.
        - Has anyone here ever tried to debug JS in IE? The error messages are worse than useless; they only give you a line number, but not the name of the file the error occurred in. Not too useful. The second utility is the Microsoft Script Debugger, which helps you trace JavaScript errors to the file and lines the error occurred in.



Here are some more detailed discussions of the issues I've addressed today. I encourage you to read them at your leisure.



Until very recently, markup was treated as something of an afterthought during web application development. With the increased focus on user experience in recent years, however, that has changed a bit, and software organizations are taking more care with markup and UI design in general. The most successful web applications give markup as much care and scrutiny as the back-end code receives.

Conclusion

- Successful web applications treat markup with as much care as the back-end code.
- A little time now will save you lots later.

As with software process, taking a little more time initially to get markup right will have big payoffs in the end.

Conclusion

- Successful web applications treat markup with as much care as the back-end code.
- A little time now will save you lots later.
- Questions?