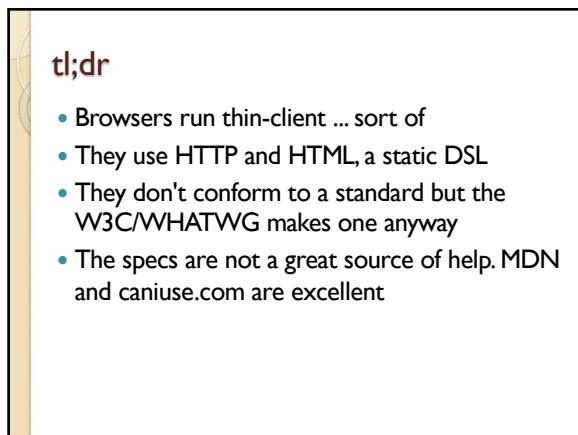
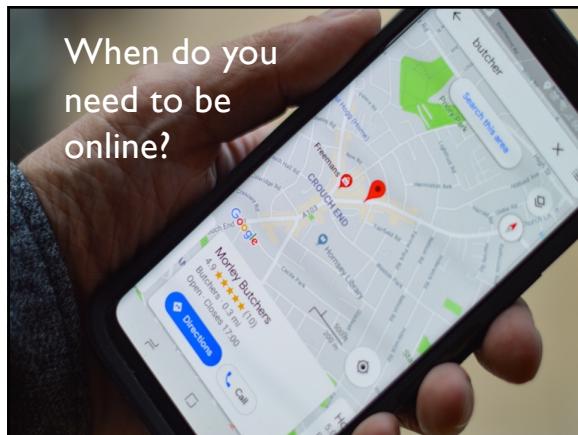


1

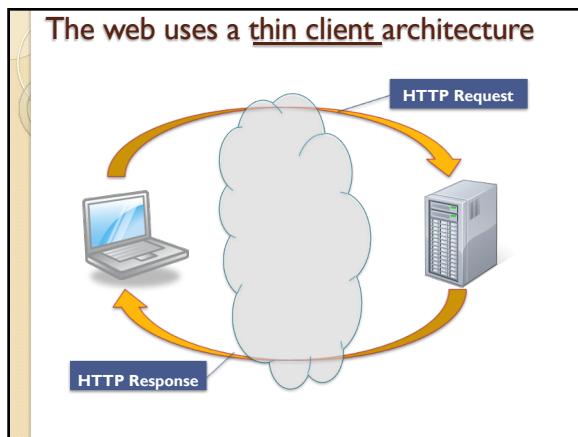


- Browsers run thin-client ... sort of
- They use HTTP and HTML, a static DSL
- They don't conform to a standard but the W3C/WWHATWG makes one anyway
- The specs are not a great source of help. MDN and caniuse.com are excellent

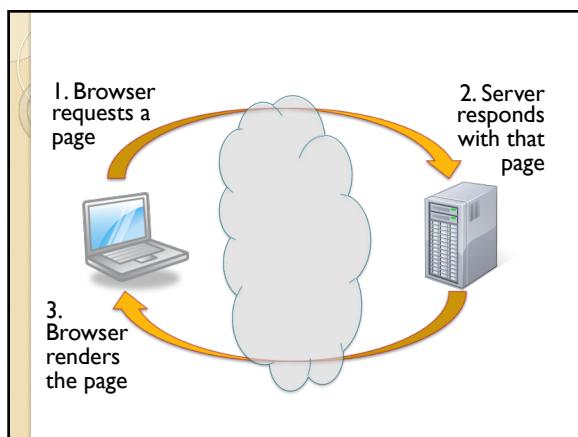
3



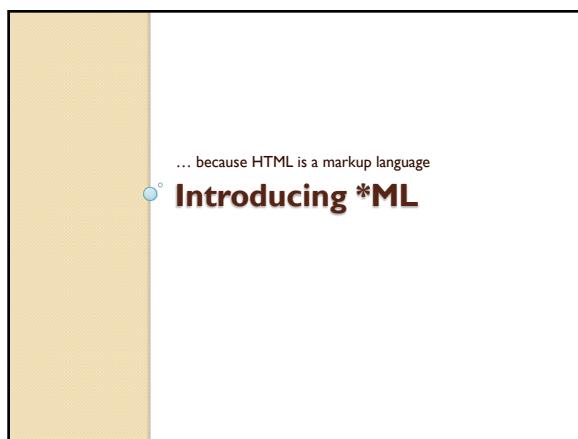
5



6



7



9

Markup languages separate annotations from data

- Usually through "tags"
- ```
<firstName>Chris</firstName>
```
- Descriptive markup
    - What kind/type/domain of data is in the tag
  - Procedural markup
    - A subroutine or procedure to run
  - Presentation markup
    - How to render the data

11

---



---



---



---



---



---



---



---

Document: Bungler OED At: "<entry>"

```

<entry>
 <hwsec>
 <hwgrp>
 <hwitem>bungler</hwitem>
 <pron>b</pron><pron>inglər</pron>. </pron>
 <etym>Also <v1>b</v1> <v2>hangler</v2>, </etym>
 <etym>t. as prec. + <xref><xlem>-ER</xlem></xref>
 <sen>One who bungles; a clumsy unskillful
 <quot>
 <qdat>1533 </qdat>
 <auth>More </auth>
 <wk>aww. Poyson. 3K. </wk>Ms. (1557)
 <txt>He is even but a very bungler.
 </sen>

```

**SGML is the original standard**

- HTML and XML conform to it
- Pioneered the idea of tags demarcating data

12

---



---



---



---



---



---



---



---

## HTML is the language of the web

- Regular UTF-8/ASCII characters sent from the server to the browser
- The browsers are written to know how to render the HTML

13

---



---



---



---



---



---



---



---



# XML

XML is a form of SGML that is used to communicate data between systems

It needs to be

- Well-formed
- Valid

---

---

---

---

---

---

14



Well-formed XML has many rules like ...

- It has a single root element
- All tags are closed or self-closing
- Tags can nest, but not overlap
- Tags are alphanumeric and case-sensitive
- Attributes are quoted

---

---

---

---

---

---

15



HTML is not XML but should follow most rules of well-formedness

- Okay  

```
<div>Best. Company. <style="color: red;">Ever</style>!</div>
```
- Not okay  

```
<div>Best. Company. <style="color: red;">Ever</div></style>!
```



---

---

---

---

---

---

16



Different browsers lead to different renderings

20

---



---



---



---



---



---

How do the browser manufacturers know what to draw?



**World Wide Web Consortium**



**Web Hypertext App Tech Working Group**

W3C and WHATWG publish standards  
No one is required to comply

22

---



---



---



---



---



---



---

How a specification is created

1. Developers unofficially come to a consensus
2. Someone ships code that works
3. Other browsers implement that same feature
4. The W3C & WHATWG make it official
5. All browsers eventually support it

23

---



---



---



---



---



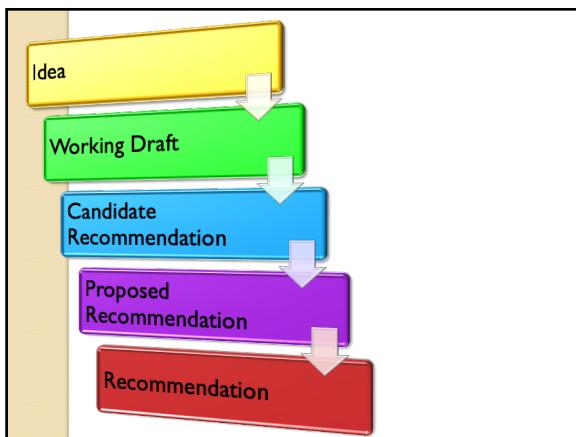
---



---



---



24

---



---



---



---



---



---



---



25

---



---



---



---



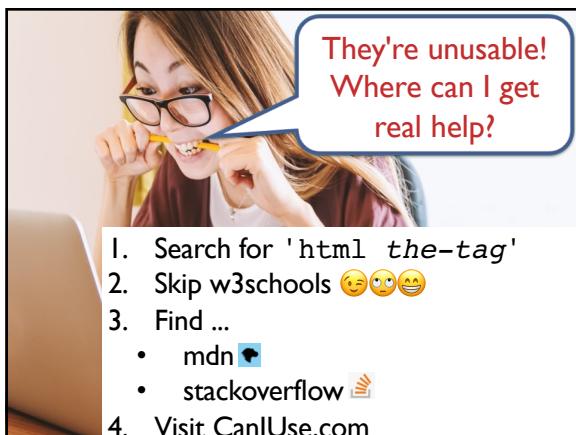
---



---



---



26

---



---



---



---



---



---



---

 tl;dr

- Browsers run thin-client ... sort of
- They use HTTP and HTML, a static DSL
- They don't conform to a standard but the W3C/WWHATWG makes one anyway
- The specs are not a great source of help. MDN and caniuse.com are excellent

---

---

---

---

---

---

---

33

 ⚒ Page Setup

The subtleties of correct HTML!

---

---

---

---

---

---

---

35

```
<h1>HTML is the language of the web</h1>
<p>It conforms to certain rules</p>

 Tags may nest but not overlap
 Tags must be closed or self-closing
 They may have attributes which ...

 may have quoted values
 are usually kebab-cased
 <li class="attributes alter">


```

---

---

---

---

---

---

---

38

Laying out a document

39

---

---

---

---

---

---

Good HTML  
has four  
elements

- DOCTYPE
- html
- head
- body

40

---

---

---

---

---

---

The DOCTYPE declaration

- This DOCTYPE says to draw the page *normally*  
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
- Here's the one for HTML5  
<!DOCTYPE html>

41

---

---

---

---

---

---



42

---

---

---

---

---

---



43

---

---

---

---

---

---



45

---

---

---

---

---

---

Altogether it may look like this

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

46

---



---



---



---



---



---



---



---

Commenting your  
HTML

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

48

---



---



---



---



---



---



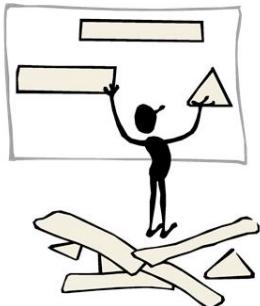
---



---

There are many other tags for layouts

- We'll get to these in a few chapters.
- nav
- article
- aside
- header
- footer
- main



50

---



---



---



---



---



---



---



---



## Displaying text

51

---

---

---

---

---

---



### The <p> tag is for phrasing content (like text)

- <p> ... </p>
- Creates a line break before and after

52

---

---

---

---

---

---



### <br /> creates a line break

- It is a singleton tag

```
<p>
Kreese: Sweep the leg.

[Johnny stares at him in shock]

Kreese: Do you have a problem with that?

Johnny Lawrence: No, Sensei.

Kreese: No mercy.
</p>
```

53

---

---

---

---

---

---

## Headings can go from levels 1 to 6

```
<head>
<title>Greendale Classes</title>
</head>
<body>
<header>Choose your classes</header>
<h1>Class List</h1>
<h2>College of Business</h2>
<h3>Finance Department</h3>
<h4>Undergraduate classes</h4>
<h5>Business Valuations 101</h5>
<h6>Section 114</h6>
```

54

---

---

---

---

---

---

## Preformatted text

- HTML usually collapses whitespace
- <pre> preserves it

```
<pre>

 \ /
 \ /
 \/
</pre>
```

**Without <pre>**



**With <pre>**



55

---

---

---

---

---

---

## There are special characters

&ampnbsp  
&copy;  
&lt;  
&gt;  
&amp;  
&reg;  
&deg;

© < > & ® °

57

---

---

---

---

---

---

## We can create two types of lists

- Unordered
  - Usually bulleted  
 <ul> ... </ul>
  - Jeff Winger
  - Britta Perry
  - Abed Nadir
  - Shirley Bennett
  - Annie Edison
  - Troy Barnes
  - Pierce Hawthorne
- Ordered
  - With numbers or letters  
 <ol> ... </ol>
  - 1. Jeff Winger
  - 2. Britta Perry
  - 3. Abed Nadir
  - 4. Shirley Bennett
  - 5. Annie Edison
  - 6. Troy Barnes
  - 7. Pierce Hawthorne
- Both use list items  
<li> ... </li>

58

## Have different types of ordered lists

- ```
<ol style="list-style-type: _____">
```
- decimal (default)
 - upper-alpha
 - lower-alpha
 - upper-roman
 - lower-roman
 - ... and more

59

You can have different bullets with unordered lists

- Shapes
 - list-style-type: none;
 - disc
 - square
 - circle
- Any picture
 - list-style-image: url('stewie.jpg');



60

Just a note on Unicode

- UTF-8 can handle most real-world character sets like ...
- Arabic
- Cyrillic
- Hebrew
- Kanji
- Etc.

No Klingon???

68

Displaying links

69

The anchor tag allows hyperlinking

- Allows linking from section to section and page to page.
- Link to another page:
`Go to other site`
- Link to someplace on this page:
`Go to another paragraph.`
- ... and ...
``

70

But anchor is more capable than you think!

- You can email:

```
<a href="mailto:rap@creator.net">  
Email Rap</a>
```

- You can call:

```
<a href="tel:8675309">Call Jenny</a>
```

- You can even text:

```
<a href="sms:8675309?body=You up?">  
Text Jenny</a>
```

71

Displaying images

73

Add images with

```

```

An actual file

Required for ally

How to resize*



There's a much cleaner
way to resize later.

- To make it a link, wrap it in an <a> tag:

```
<a href="tic.com"></a>
```

74

New! Lazy Loading

```

```

- Chrome will check the 1st 2 Kb of the image for size in order to put placeholders.
- Will defer loading until it has idle time.

75

How to handle the alt attribute

1. Always include the alt attribute
2. alt="" if the image is just eye candy
3. Images of text: alt should be the text
4. Keep it very short
 - guy in leather jacket with boots on table
 - woman late for class
 - man in apron preparing a baloney sandwich
5. Don't say it's an image or photo.

76

The title attribute

```
<any title="Here's a tooltip" />
```

- Generally avoid it.



77



tl;dr

- The basic layout of a page contains exactly one each of a DOCTYPE declaration, a html tag, a head tag, and a body tag
 - You'll also need paragraph tags to wrap text, line breaks, and probably anchor tags
 - Certain tags are used primarily for text like <p>
 - Headings do more than just print big text
 - <pre> tags allow preformatted text
 - Special characters can be added with &
 - Lists are done with , and
-
-
-
-
-
-

79



◦ Debugging your HTML and CSS



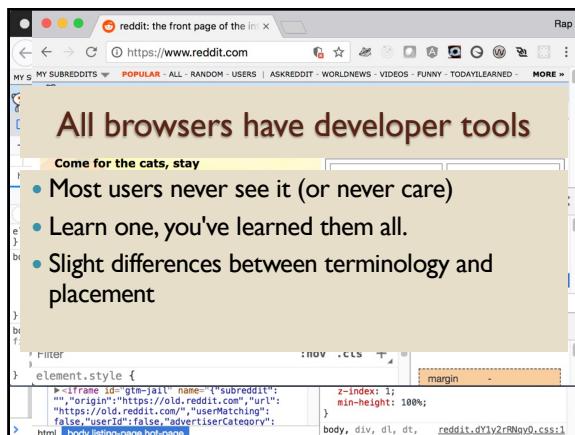
tl;dr

- Debugging is available but it must be done in the browser
 - All browsers have their own debugging tools
 - Fortunately they all behave pretty much the same way
 - You can modify your HTML and CSS, examine the HTTP traffic, and simulate a mobile device
-
-
-
-
-
-

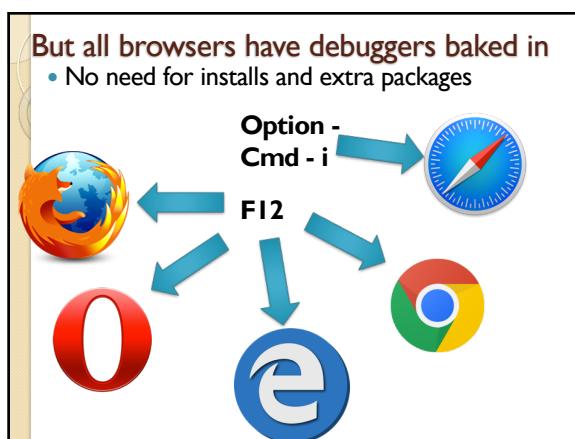
85



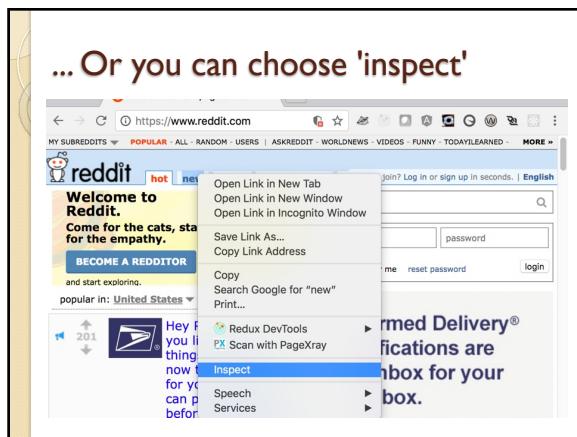
86



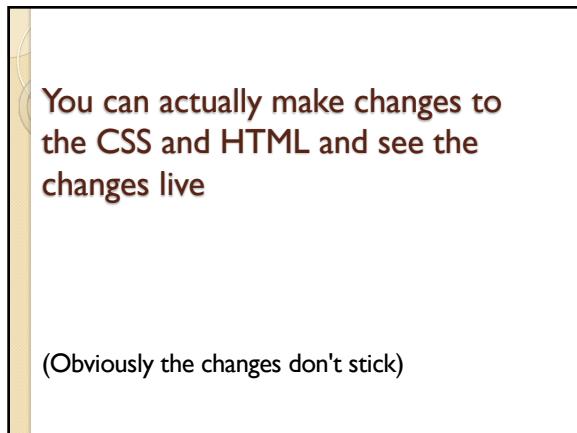
87



88



89



90



91

The screenshot shows the DevTools Elements tab with the DOM structure of a movie page from IMDB. The 'Computed' tab is selected in the sidebar. A specific table row is highlighted, and its computed styles are displayed in the main panel. The styles include margin, border, padding, width, height, and font properties.

Working with CSS in the Computed tab

92

The screenshot shows the DevTools Network tab for an Amazon page. It displays a waterfall chart at the top showing the duration of each request. Below the chart is a table listing 256 network requests. The columns include Name, Status, Type, Initiator, Size, Time, and Waterfall. The table shows various resources like stylesheets, images, and fonts being loaded.

Examine traffic on the Network tab

Shows each resource requested, its HTTP method, result, type, and speed

93

The screenshot shows the DevTools Elements tab for a Walmart mobile device simulation page. The 'Computed' tab is selected. A specific element is highlighted, and its computed styles are shown in the main panel. The styles include element.style, body, and html properties.

94

Then pick your preferred device

Choose your device under the Emulation tab

The screenshot shows the developer tools interface with the 'Emulation' tab selected. Under the 'Device' section, 'Model' is set to 'Apple iPhone 6 Plus'. Other options like 'Emulate screen resolution' (Resolution: 414x736), 'Device pixel ratio' (3), and checkboxes for 'Emulate mobile' and 'Shrink to fit' are checked. A note at the bottom says: '⚠ You might need to reload the page for proper user agent spoofing and viewport rendering.'

95

How the Page Looks on a Mobile Device

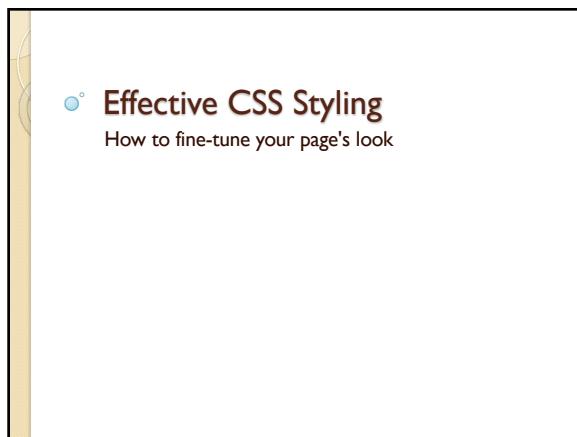
The screenshot shows a mobile browser view of a flight search results page. On the left, a note says: 'Note: you're still on the desktop browser. It just looks like a mobile browser.' On the right, there are four buttons with arrows pointing to them: 'Simulate a slow network', 'Choose your mobile browser agent', and 'Rotate the device'. The browser's developer tools are visible at the top.

96

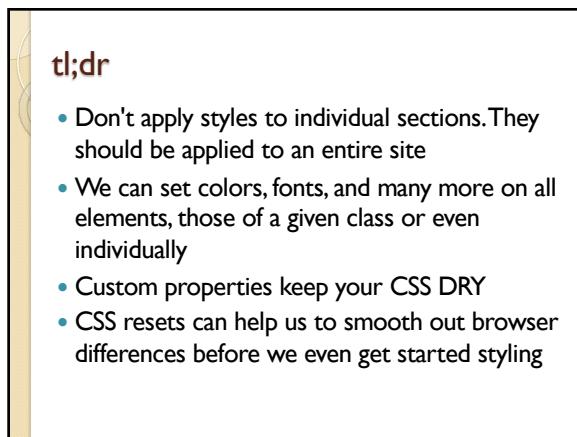
tl;dr

- Debugging is available but it must be done in the browser
- All browsers have their own debugging tools
- Fortunately they all behave pretty much the same way
- You can modify your HTML and CSS, examine the HTTP traffic, and simulate a mobile device

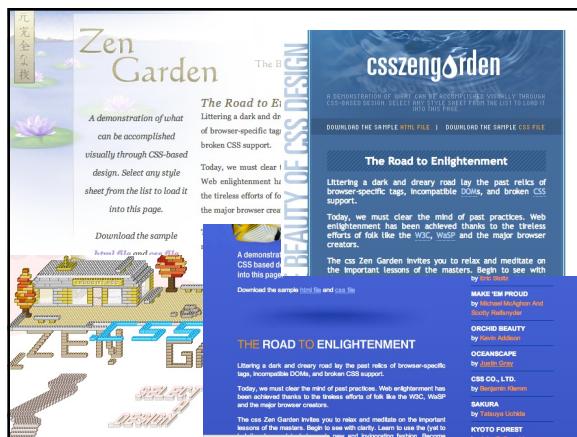
97



99



101



102

Old-school HTML formatting was a nightmare to maintain

- This worked:

```
<p><font color="red"><b><i>
All your base are belong to us!
</i></b></font></p>
```

- Nigh impossible to maintain.

All your base are belong to us!

103

Styles define a look and feel

Can be set on ...

1. a particular element,
2. a particular page, or
3. the entire site

```
.alert {
  color: #ff4451;
  font-weight: bolder;
  padding: 1px;
```

104

`<p style="color: red;">`

You can set a style for a single element ...



105

... Or on the page ...

```
<head>
<title>I &lt;3 Styles!</title>
<style type="text/css">
.headlines, .sublines, .infotext {
  font-face: arial;
  color: black;
  background: yellow;
  font-weight: bold;
}
.headlines {font-size:14pt;}
.sublines {font-size:12pt;}
.infotext {font-size: 10pt;}
</style>
</head>
```

106

... or for the entire site

```
<link rel="stylesheet" href="site.css" />
```

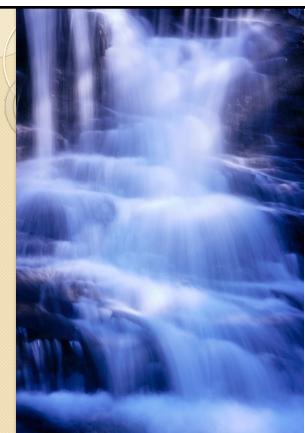


108

The styles cascade

All styles flow from the entire site to all pages, sections, and elements

- unless they're overridden at the lower level
- lowest one wins



109

The styles nest

When you set the style on something higher up in the DOM, it takes effect for everything within that element
(aka. "style inheritance")



110

The syntax for any style is this ...

The thing(s) on the page(s) we're applying the style to

```
selector {
    style: value;
    style: value;
    ...
}
```

A listing of the styles that we want to apply

112

The selectors come in many flavors

- For all elements of a type
 - `div { ... }`
 - `p { ... }`
 - `li { ... }`
- By class
 - `.alert { ... }`
 - `.finePrint { ... }`
- By ID
 - `#goButton { ... }`
 - `#errorDiv { ... }`
- Many, many more

113

When styles collide, who wins?

1. Directly targeted styles beat inherited styles
2. More specific beats less specific
 - #id beats .class
 - .class beats <element>
3. Last one read beats the first

114

This is !important

- You can prevent overriding with *!important*
- ```
div {
 background-color: white !important;
}
```



Use with caution! Makes it  
tough to debug.

115

---

---

---

---

---

---

---

... like variables in your CSS

## CSS custom properties

116

---

---

---

---

---

---

---

**Problem:** Having the same value in many places in CSS is hard to maintain

```
.class1 {
 color: #bb4ef2;
}
.class2 {
 background-color: #bb4ef2;
}
input {
 box-shadow: 5px 5px #bb4ef2;
}
```

117

---



---



---



---



---



---



---

**Solution:** Create CSS variables!



Must begin with '--'  
Referenced with 'var(...)'

```
:root { --logo-color: #bb4ef2; }
.class1 {
 color: var(--logo-color);
}
.class2 {
 background-color: var(--logo-color);
}
input {
 box-shadow: 5px 5px var(--logo-color);
}
```

118

---



---



---



---



---



---



---



---

Resets

119

---



---



---



---



---



---



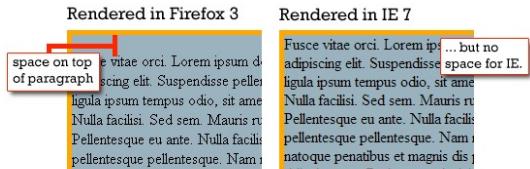
---



---

## A plain, unstyled page looks different in different browsers

- Why?



120

## To make all pages look the same, we use CSS resets

```
html, body, div, span, object, iframe, h1, h2, h3,
h4, h5, h6, p, blockquote, pre, abbr, address, cite,
code, del, dfn, em, img, ins, kbd, q, samp, small,
strong, sub, sup, var, b, i, dl, dt, dd, ol, ul, li,
fieldset, form, label, legend, table, caption,
tbody, tfoot, thead, tr, th, td, article, aside,
canvas, details, figcaption, figure, footer, header,
hgroup, menu, nav, section, summary, time, mark,
audio, video {
 margin:0;
 padding:0;
 border:0;
 outline:0;
 font-size:100%;
 vertical-align:baseline;
 background:transparent;
}
```

121

## You can get good resets from various places around the Internet

- Here are a few:
- <http://meyerweb.com/eric/tools/css/reset/>
  - <http://code.google.com/p/html5resetcss/>
  - <http://yui.yahooapis.com/3.5.1/build/cssreset/cssreset-min.css>

122

**tl;dr**

- Don't apply styles to individual sections. They should be applied to an entire site
- We can set colors, fonts, and many more on all elements, those of a given class or even individually
- Custom properties keep your CSS DRY
- CSS resets can help us to smooth out browser differences before we even get started styling

123

---

---

---

---

---

---

---

**Semantic Grouping**

So they can be positioned and styled as a unit

128

---

---

---

---

---

---

---

We need groupings to specify areas of our pages.



129

---

---

---

---

---

---

---

```
<div id="mw-head">
...
</div>
<div id="mw-panel">
<div id="p-navigation">
</div>
<div id="p-interaction">
</div>
</div>
<div id="mw-body">
<div id="first-heading">
</div>
<div id="bodyContent">
</div>
<div class="infobox">
</div>
</div>
```

**HTML has several elements for grouping**

130

---

---

---

---

---

---

---

**Elements come in two flavors: block-level and inline**

**Block-level**

- Rectangular objects
- Have heights, widths, & margins
- Line breaks before and after
- `<div>, <p>, <h2>, <ol>, <ul>, <hr>`

**Inline**

- Part of the flow of document text
- No concept of width or height
- `<span>, <a>, <img>`

131

---

---

---

---

---

---

---

**Spans allow us to group things without disrupting the flow of text before and after**

`<p>This is a <span class="alert">warning</span> so consider yourself warned.</p>`



132

---

---

---

---

---

---

---

## HTML5 semantic elements communicate meaning

- ⌚ section
- ⌚ details
- ⌚ nav
- ⌚ summary
- ⌚ article
- ⌚ figure
- ⌚ aside
- ⌚ hr
- ⌚ header
- ⌚ footer
- ⌚ main

133

---

---

---

---

---

---

---



134

---

---

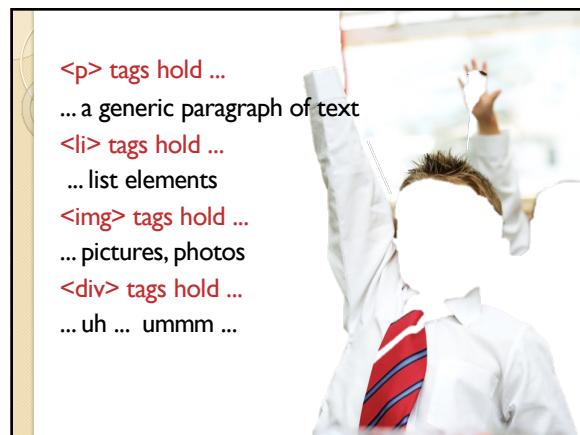
---

---

---

---

---



135

---

---

---

---

---

---

---

## Semantic elements add meaning to HTML

```

<div id="mw-head"> <header>
...
</div> ...
<div id="mw-panel"> <section>
...
</div> <nav>
<div id="p-navigation"> </nav>
</div> ...
<div id="p-interaction"> <nav id="inter">
</div> </div>
</div> ...
<div id="mw-body"> <main>
...
<div id="first-heading"> <header>
</div> ...
<div id="bodyContent"> <article>
...
<div class="infobox"> <section id="infobox">
...
</div> </section>
</div> ...

```

136

## The W3C spec says the section element

...

"... represents a generic document or application section. A section, in this context, is a thematic grouping of content, typically with a heading."

Huh!?

137

## A section is the place to put general text in your document

Basically any grouping of things that conceptually belong together

- Chapters
- Each tabbed pages in a tabbed dialog box
- Sections of a thesis
- A web site's home page could be split into sections for an introduction, news items, contact information

138

## A section sounds like a div, right?

### <section>

- Generally would appear in an outline of a page

### <div>

- Has no semantic meaning
- Mostly for grouping things (like to apply a class or a script to it)
- Use as a last resort – only when nothing else fits

139

---



---



---



---



---



---



---



---

## Articles are for content that could be syndicated

- User-submitted comments
- Newspaper articles
- Magazine articles
- Blog entries
- Forum posts
- etc.



140

---



---



---



---



---



---



---



---

## The differences between sections and articles are subtle

Use an article if ...	Use a section if ...
It is self-contained	It is <b>part</b> of a bigger thing
It stands alone	It seems to require other parts to make it complete
Someone might read it over a cup of coffee	"Why anyone just sit and read this?"
It might make sense to republish it on another web page	No one would republish it because it only has value on your site
It would have a title	It would have a heading

141

---



---



---



---



---



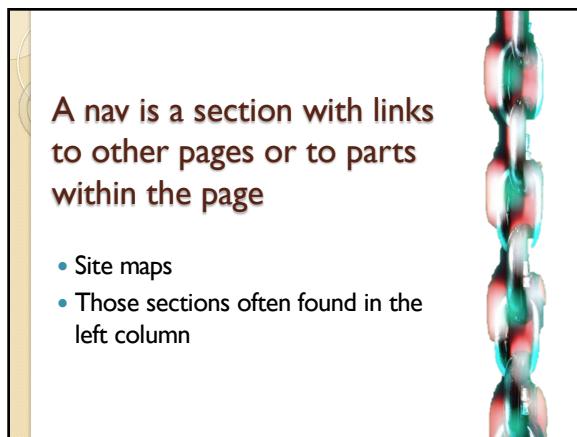
---



---



---



142

---

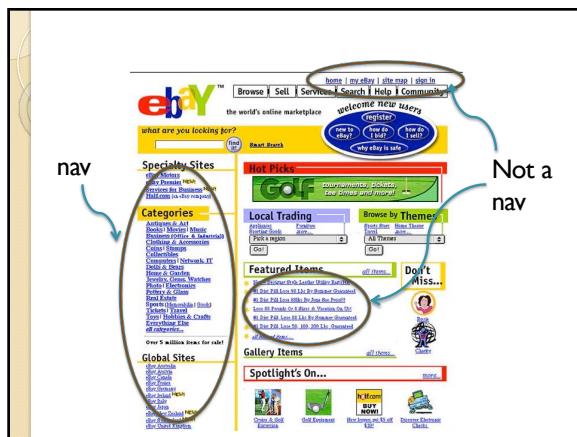
---

---

---

---

---



143

---

---

---

---

---

---



146

---

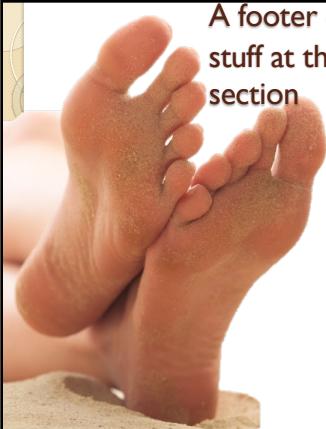
---

---

---

---

---



**A footer contains the stuff at the bottom of a section**

- About the author
- Company data
- Links
- Date it was written
- Copyright notice

147

---



---



---



---



---



---



---



---



---



---



**<main>**

- Newest of the new tags
- 1 per doc
- Must not be a descendent of an `<article>`, `<footer>`, `<header>`, or `<nav>` element.

```

<html>
 <body>
 <header>
 </header>
 <main>
 Main stuff goes here
 </main>
 <footer>
 </footer>
 </body>
</html>

```

148

---



---



---



---



---



---



---



---



---



---



---



**<details> and <summary> allow collapsible regions without JavaScript**

```

<details>
 <summary>Theme song</summary>
 <pre>
 104 days of summer vacation
 Then school comes along just to end it
 So the annual problem for our generation
 Is finding a good way to spend it
 </pre>
</details>

 ▶ Theme song

```

▼ Theme song

```

 104 days of summer vacation
 Then school comes along just to end it
 So the annual problem for our generation
 Is finding a good way to spend it

```

149

---



---



---



---



---



---



---



---



---



---



---

### To change the arrow image on <details>

```
summary::-webkit-details-marker {
 display: none
}

summary:after {
 background: url(some-picture);
 float: left;
 height: 20px;
 width: 20px;
 content: " ";
}

details[open] summary:after {
 background: url(other-picture);
}
```

150

---



---



---



---



---



---



---



---



---



---



---



---

 **Do this ...**

```
<figure>

 <figcaption>
 Perry (Agent P)
 </figcaption>
</figure>
```

**... not this**

```
<div class="figure">

 <p>
 Perry (Agent P)
 </p>
</div>
```

151

---



---



---



---



---



---



---



---



---



---



---



---

### tl;dr

- HTML has provided us with inline spans and block divs to group things for years now
- But now we have semantic elements that will help us with better organization, abstraction, and SEO

155

---



---



---



---



---



---



---



---



---



---



---



---



## How to Position with CSS

The box model, overflow, absolute, relative, and fixed positioning

158

---

---

---

---

---

---



### tl;dr

- This chapter is all about putting things on a page in relation to other things
- The box model helps in the aesthetic layout of our sites
- centering
- position: absolute/relative/fixed/static

160

---

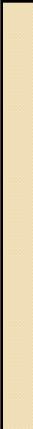
---

---

---

---

---



Laying out sections of a page

## The box model

161

---

---

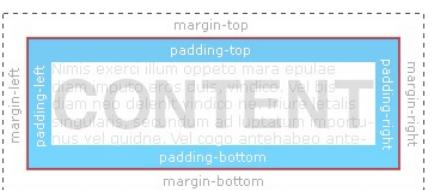
---

---

---

---

## The box model provides aesthetic space



**Margin**

- Space outside the container between sections

**Padding**

- Space inside the container between the container and the contents

The diagram illustrates the box model with three nested boxes. The outermost dashed box represents the margin. The innermost solid blue box represents the padding. Between these two is the content area. Labels include: margin-top, padding-top, padding-bottom, margin-bottom, margin-left, padding-left, and margin-right.

162

---



---



---



---



---



---



---



---



---

## There are many units of measure

- Pixels (px)
  - Affected by screen resolution
  - Great for absolute control over layout
- Ems (em)
  - Proportion of the default font of the browser
  - Great for accessibility
- Percentages (%)
- Proportion of the width or height of the container
- Viewport size (vh and vw)
  - Like % but better

163

---



---



---



---



---



---



---



---



---

## So, which do I use?

- Use points only for print views
- Use percentages and pixels for screen layout
- Use ems for text
- Best of all worlds this way!

166

---



---



---



---



---



---



---



---



---

**You can have different box sides**

- The four sides are specified in "TRouBLE" order.
- Top, Right, Bottom, Left

Top, Right, Bottom, Left

169

---

---

---

---

---

---

---

---

Using shorthands can save characters

170

---

---

---

---

---

---

---

---

Centering

176

---

---

---

---

---

---

---

---

## Horizontal centering

- `text-align: center;`
- only work on inline elements and not block elements
- `margin: 0px auto;`
- The auto means make it the same on both sides.

178

---



---



---



---



---



---



---



---

## Vertical centering

- `vertical-align: top | middle | bottom | et. al.`
- You apply it to the element inside. The one you want centered, not on the parent.
- It only works on a `<p>` if it is `display:inline-block;`
- Kind of picky.

179

---



---



---



---



---



---



---



---

## Centering using flex

This method is a little inelegant, but it is easy and works almost everywhere

```
div {
 display: flex;
 justify-content: center; /* vertical */
 align-items: center; /* horizontal */
}
```



Much more about flexbox later in the course. This'll become clearer then.

180

---



---



---



---



---



---



---



---

**The position style**

181

---

---

---

---

---

---

---

**The position style has discrete values:**

**static**  
**relative**  
**absolute**  
**fixed**

(and a few others)

182

---

---

---

---

---

---

---

- The elements on the page flow.
- Inline elements live side-by-side as wide as their container allows. Then they go down to the next line.

**static (the default)**

```
div:nth-child(2) {
 position: static;
}
```

183

---

---

---

---

---

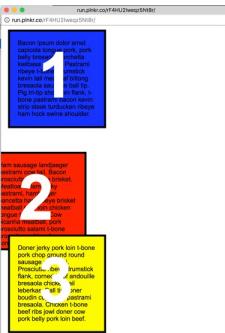
---

---

- It participates in the layout of the page as normal but then is moved relative to where it would have been.
- Positioned relative to its first positioned parent

```
div:nth-child(2) {
 position: relative;
 top: 50px;
 right: 50px;
}
```

## relative

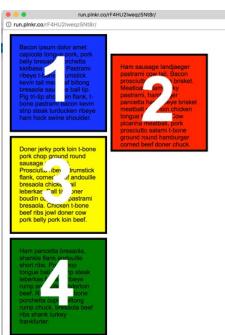


184

- Takes it out of the document flow.
- Positions it absolutely on the page.

```
div:nth-child(2) {
 position: absolute;
 top: 50px;
 right: 50px;
}
```

## absolute



185

- It will be fixed to a spot in the browser and **not** the page.
- It ignores everything else on the page.

```
div:nth-child(2) {
 position:fixed;
 top: 0px;
 right: 0px;
}
```

## fixed



186



### tl;dr

- This chapter is all about putting things on a page in relation to other things
- The box model helps in the aesthetic layout of our sites
- centering
- position: absolute/relative/fixed/static

---

---

---

---

---

---

---

188



### ➊ Page Layout Strategy

What are my options and which do I choose?

---

---

---

---

---

---

---

191



### tl;dr

- Pages have different sections which require some planning to lay them out
- Our options:
  - Tables and absolute positioning are inflexible
  - Floating
  - inline-block
  - flexbox
  - grid
- But how do I know which to use?

---

---

---

---

---

---

---

194

**How do you layout pages?**

1. Define sections
2. Define sizes
3. Get them to live side-by-side

195

---

---

---

---

---

---

---

---

**And how do you get things to live side-by-side?**

- Tables
- Absolute positioning
- Inline sections

196

---

---

---

---

---

---

---

---

**A word about page flow**

- Remember, we have two types of elements
- **inline elements**
  - Text and other things flow around the element
  - No concept of width or height
- **block elements**
  - A break appears before and after it
  - Has width, height, borders, padding, and margins
- Sounds like we'll need a hybrid of the two

200

---

---

---

---

---

---

---

---

## Four ways to hybrid *inline* and *block*

1. floated divs
2. display: inline-block
3. flex boxes
4. grids

201

---



---



---



---

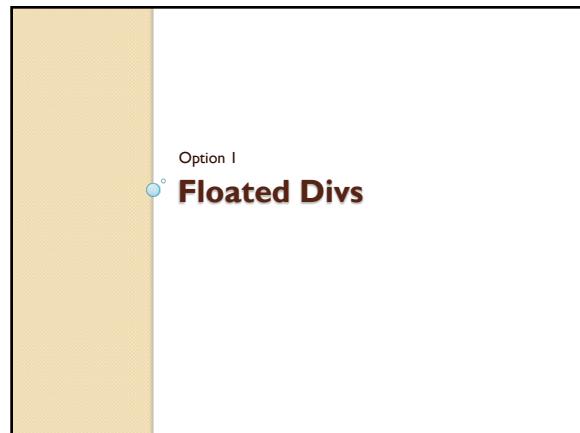


---



---

202




---



---



---



---



---



---

203

### From the CSS spec ...

A float is a box that is shifted to the left or right on the current line. The most interesting characteristic of a float (or "floated" or "floating" box) is that content may flow along its side

`float: left;`  
Says "I'm going to move left.  
Let the thing below me float up on my right"  
`float: right` does the same but moves to the right

`clear: both;`  
Says "You below can't float higher than me"  
re-establish breaks

---



---



---



---



---



---

## Floating is the weirdest thing in layouts

- A floated element allows things below it to float up next to it -- if there's room
- Floating things takes them out of the normal flow of the text.

Without float

div1  
div2  
div3  
div4  
div5

With float

div1 div2 div3 div4 div5

204

---



---



---



---



---



---



---



---



---



---



---

## Clear the last item if needed

`clear: both;`

- Says "You below can't float higher than me"
- re-establish breaks

Without clear

div1 div2 div3 div4 div5 Next paragraph

With clear

div1 div2 div3 div4 div5  
Next paragraph

205

---



---



---



---



---



---



---



---



---



---



---

## When is the time to use float?

You want block elements to live side-by-side

```
li.sideBySide {
 float: left; /* or right */
}
```



You may want to set a width because block elements are greedy horizontally

206

---



---



---



---



---



---



---



---



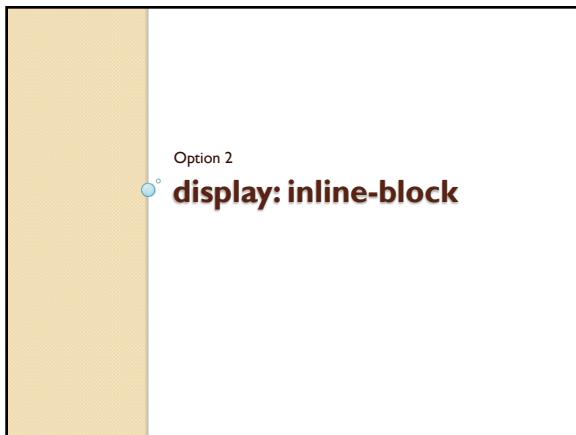
---



---



---



211

---

---

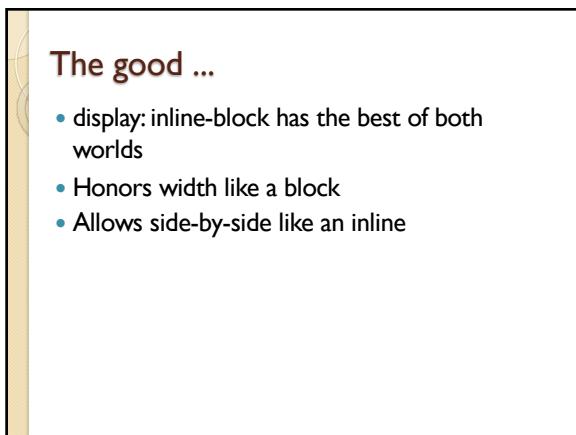
---

---

---

---

---



212

---

---

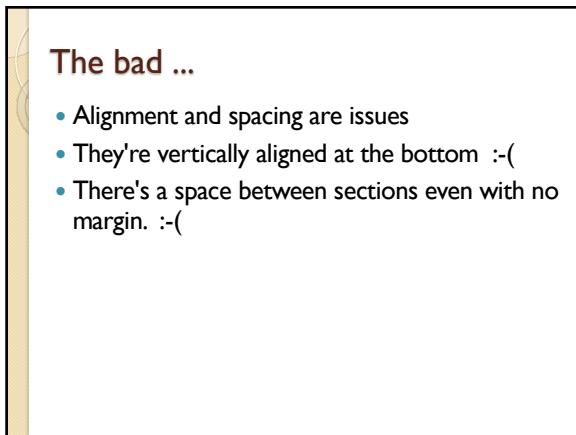
---

---

---

---

---



213

---

---

---

---

---

---

---

**The ugly ...**

```
section {
 display: inline-block;
 width: 1px;
 margin-left: 10px;
}
```

Vestibulum ante ipsum primis in faucibus orci luctus et dolore magna aliquam erat volutpat. Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusamus et iusto odio dignissimos.

**Content Section**

Aenean magna eros, pretium vitae commodo non, interdum eu metus. Nam non nisi turpis, eget adipiscing purus. Integer suscipit tempus est, non fermentum dui accumsan vitae. Vestibulum ut massa neque. Quisque a neque ut augue fermentum varius non vitae orci. Aenean sed pellentesque risus. Aenean quis nunc tortor, eu aliquet massa. In hac habitasse platea dictumst. Duis erat purus, condimentum et ullamcorper at, aliquet et urna. Fusce blandit volutpat velit in consectetur. Duis imperdiet iaculis sodales. Proin sodales hendrerit lacini.

**Links**

- Nam ullamcorper
- commodo nibh
- tincidunt congue
- Vestibulum id
- Vivamus sed

**Ads**

214

---

---

---

---

---

---

---

---

---

Suspendisse et dui dolor. Nam ac neque neque, sed mollis dolor. Etiam cursus nibh non e lobortis id condimentum tortor portitor.

**Content Section**

Aenean magna eros, pretium vitae commodo non, interdum eu metus. Nam non nisi turpis, eget adipiscing purus. Integer suscipit tempus est, non fermentum dui accumsan vitae. Vestibulum ut massa neque. Quisque a neque ut augue fermentum varius non vitae orci. Aenean sed pellentesque risus. Aenean quis nunc tortor, eu aliquet massa. In hac habitasse platea dictumst. Duis erat purus, condimentum et ullamcorper at, aliquet et urna. Fusce blandit volutpat velit in consectetur. Duis imperdiet iaculis sodales. Proin sodales hendrerit lacini.

**To fix it ...**

```
vertical-align: top;
margin: 0px -4px;
```

**Links**

- Nam ullamcorper
- commodo nibh
- tincidunt congue
- Vestibulum id
- Vivamus sed

**Ads**

215

---

---

---

---

---

---

---

---

---

° A very high-level intro to flexbox and grid

216

---

---

---

---

---

---

---

---

---

## Flexbox

- Lay out either across or down but not both.
- If you have too many things to fit, you can wrap down to the next line.
- If you don't wrap you can decide how to allocate the extra space

218

---

---

---

---

---

---

---

## Grid

- Lay out components in rows and columns simultaneously
- Like a <table> but cleaner.

219

---

---

---

---

---

---

---

So ... which one do I learn?

**float vs inline-block vs  
grid vs flexbox**

220

---

---

---

---

---

---

---

<b>float: left;</b>	You need 2 elements to be side-by-side. Not best for full pages anymore.
<b>display: inline-block;</b>	You want 3 or more page sections to be side-by-side
<b>display:flex;</b>	1-dimensional possibly responsive layouts (ie. rows <u>or</u> columns)
<b>display:grid;</b>	2d layouts (ie. rows <u>and</u> columns)

221

---



---



---



---



---



---



---

<b>Indeterminate number of items</b>
<ul style="list-style-type: none"> <li>• All of the layout options handle extra items well.</li> <li>• Flexbox can wrap and it will wrap as long as it needs to.</li> <li>• Grid will repeat the last row as many times as needed.</li> <li>• Float wraps also.</li> </ul>

224

---



---



---



---



---



---



---

<b>tl;dr</b>
<ul style="list-style-type: none"> <li>• Pages have different sections which require some planning to lay them out</li> <li>• Our options: <ul style="list-style-type: none"> <li>◦ Tables and absolute positioning are inflexible</li> <li>◦ Floating</li> <li>◦ inline-block</li> <li>◦ flexbox</li> <li>◦ grid</li> </ul> </li> <li>• But how do I know which to use?</li> </ul>

225

---



---



---



---



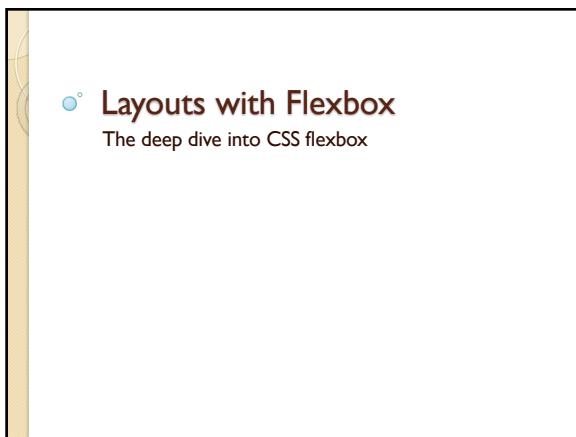
---



---



---



232

---

---

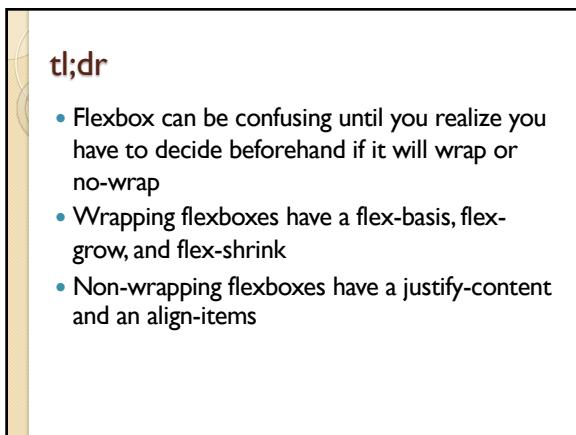
---

---

---

---

---



233

---

---

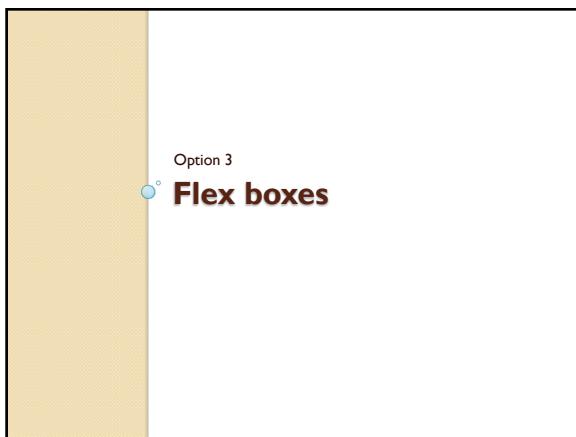
---

---

---

---

---



234

---

---

---

---

---

---

---



235

---

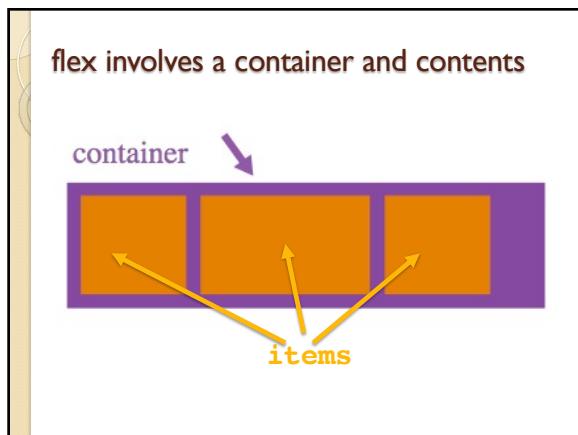
---

---

---

---

---



236

---

---

---

---

---

---

Mark the container as a flex-box

```
#container {
 display: flex;
}
```

- And best case scenario ... that's all you need!

238

---

---

---

---

---

---

The container has axes

Most of us would call these

- horizontal
- vertical

239

---

---

---

---

---

---

The container lays out the items

```
#container {
 flex-direction: column;
 flex-direction: column-reverse;
 flex-direction: row;
 flex-direction: row-reverse;
}
```

240

---

---

---

---

---

---

Are we going to wrap or not?

Two ways to think about flexbox ...

241

---

---

---

---

---

---

## Two ways to think about flex

- 1. We need it to not wrap
  - For when you have a small and fixed number
  - You want the space allocated to each to flex
  - You'll use flex-basis, flex-shrink, and flex-grow.
- 2. We may need it to wrap
  - For when you have an arbitrary number of items
  - You want to create X rows
  - You want the number of items to flex
  - You'll use justify-content, align-items

243

---



---



---



---



---



---



---



---



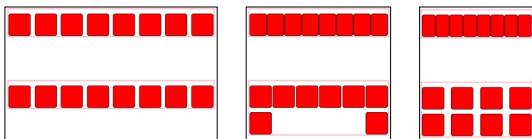
---



---

## Think of flexbox like this ...

- It's a one-dimensional layout - row or column
- But if you have too many items to fit, it can wrap to a new line - but you can prevent that.



244

---



---



---



---



---



---



---



---



---



---

## Not wrapping with flex

- We have a fixed number of items.
- If there's not enough along the main axis, we steal some from each item.
- If there's extra room left over, we allocate some to each item.

245

---



---



---



---



---



---



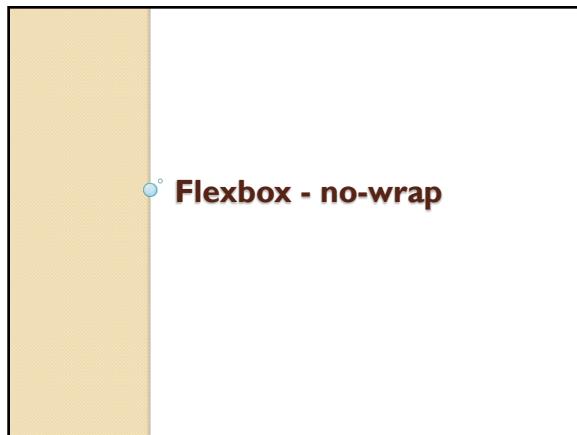
---



---



---



246

---

---

---

---

---

---

---

**flex items have sizes**

 <p>They have a "favorite" size</p> <p><b>flex-basis</b></p>	 <p>They know how to shrink from that and how to grow from that</p> <p><b>flex-shrink</b></p> <p><b>flex-grow</b></p>
------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

248

---

---

---

---

---

---

---

**Sizes of the items**

- **flex-basis** = the item's "favorite" size along the flex-direction.
  - in px, em, %, etc.
- If the viewport is increased from flex-basis ...
  - **flex-grow** = by how much it grows
    - unitless - a relative number
- if the viewport is decreased from flex-basis ...
  - **flex-shrink**= by how much it shrinks
    - unitless - a relative number

249

---

---

---

---

---

---

---

To have relative widths/heights set flex-grow/flex-shrink on each item

```
#item1 {
 flex-grow: 3;
}
#item2 {
 flex-grow: 1;
}
#item3 {
 flex-grow: 2;
}
```

- These numbers are unitless and relative to one another.

251

---

---

---

---

---

---

---

order

- Default: The order they're in on the page
- Numerically otherwise

252

---

---

---

---

---

---

---

Wrapping with flex

253

---

---

---

---

---

---

---

flex-wrap tells them to wrap when needed

```
#container {
 flex-wrap: wrap;
}
```

254

---



---



---



---



---



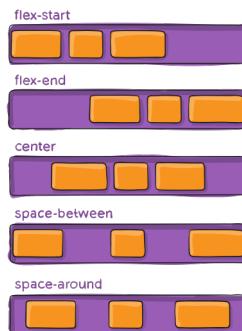
---



---

```
#container {
 justify-content: _____;
```

And the container controls the spacing



255

---



---



---



---



---



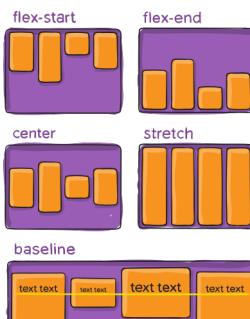
---



---

```
#container {
 align-items: _____;
```

And the container controls the spacing



256

---



---



---



---



---



---



---

### Bonus! How to center anything

```
#container {
 display: flex;
 justify-content: center;
 align-content: center;
}
```

257

---

---

---

---

---

---

---

### tl;dr

- Flexbox can be confusing until you realize you have to decide beforehand if it will wrap or no-wrap
- Wrapping flexboxes have a flex-basis, flex-grow, and flex-shrink
- Non-wrapping flexboxes have a justify-content and an align-items

260

---

---

---

---

---

---

---

### ➊ Layouts with Grid

The deep dive into CSS grids

265

---

---

---

---

---

---

---

**tl;dr**

- Pages have different sections which require some planning to lay them out
- Grids allow us to lay out pages in grids and columns by assigning lines between them to create the cells
- You can even combine the cells to create sections and assign items to the sections.

266

---



---



---



---



---



---



---



---

**First, some CSS grid concepts ...**

- *containers* have *items*
- containers have *lines* that define rows and columns (aka *tracks*)
- *cells* are where the tracks intersect
- cells can be combined into rectangular *areas*

268

---



---



---



---



---



---



---



---

**Just like in flexbox, use a container**

```
.container {
 display: grid;
 grid-template-columns: 1fr 1fr;
}
```

- A container can be anything in your document:
  - <body>
  - <div>
  - <section>
  - <whatever> (Disclaimer: not a real thing).

269

---



---



---



---



---



---



---



---

## Wait, what's that "fr" thing?

- "Free space" = what is left over after painting.
- They're distributed according to the proportional numbers of the columns.
- Why not just use %?
- Because fr takes into account grid gaps. % does not. If there are no grid gaps, they have the same effect.

270

---



---



---



---



---



---



---



---

## Everything in a grid container is by definition a grid item

- A grid will place all of its direct children
- A grid will only place its direct children

271

---



---



---



---



---



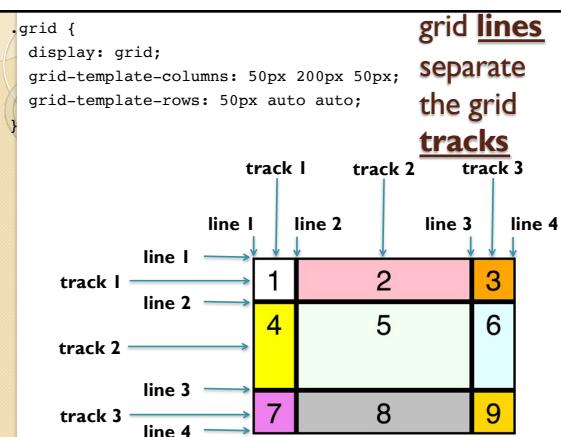
---



---



---



272

---



---



---



---



---



---

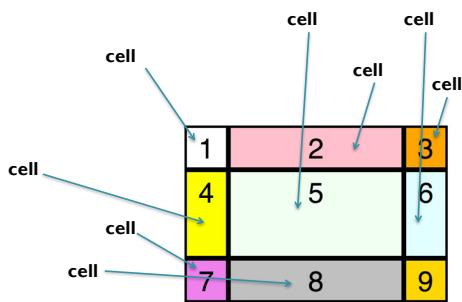


---



---

And of course grid **cells** are where the tracks (rows and columns) intersect



273

---

---

---

---

---

---

---

---

---

We've talked about fixed widths, what about fixed heights?

- You should usually let heights change based on their contents. They should be lazy -- only as tall as they must be to accommodate their contents.
- Do this by using "auto" as the height for the row.
- % and fr are meaningless in heights unless the container has a fixed size.

---

---

---

---

---

---

---

---

---

274

```
.grid {
 display: grid;
 grid-template-columns: 50px 100px 50px;
 grid-template-rows: 50px auto auto;
}
#one {
 grid-column: 1/3;
 grid-row: 1/2;
}
#two {
 grid-column: 2/4;
 grid-row: 3/4;
}
#three {
 grid-column: 2/3;
 grid-row: 2/3;
}
```

**To assign to cells, place between lines**

Fill the columns between line 1 and line 3  
Fill the rows between line 1 and line 2

---

---

---

---

---

---

---

---

---

275

**grid areas are optional groups of cells**

```
.grid {
 display: grid;
 grid-template-columns: 50px 100px 50px;
 grid-template-rows: 50px auto auto;
 grid-template-areas:
 'head head head'
 'ad content content'
 'ad content content'
}
```

276

---

---

---

---

---

---

---

---

---

**To assign to areas**

```
.grid {
 display: grid;
 grid-template-columns: 50px 100px 50px;
 grid-template-rows: 50px auto auto;
 grid-template-areas:
 'head head head'
 'ad content content'
 'ad content content'
}
#one {
 grid-area: head;
}
#two {
 grid-area: ad;
}
#three {
 grid-area: content;
}
```

277

---

---

---

---

---

---

---

---

---

**tl;dr**

- Pages have different sections which require some planning to lay them out
- Grids allow us to lay out pages in grids and columns by assigning lines between them to create the cells
- You can even combine the cells to create sections and assign items to the sections.

281

---

---

---

---

---

---

---

---

---



## Progressive Web Apps and Responsive Design

Making web sites look good on any sized screen,  
especially mobile devices

---

---

---

---

---

---

286



### tl;dr

- PWAs have tons of requirements but they allow the use of our web apps offline.
- Media queries allow us to apply different styles to different media and different screen sizes
- We can use these to have responsive web designs that flex when browsed on different devices

---

---

---

---

---

---

287



## Programming for mobile devices is difficult

<ul style="list-style-type: none"><li>• iOS</li><li>• Objective-C</li><li>• Cocoa Touch</li><li>• XCode IDE</li></ul>	<ul style="list-style-type: none"><li>• Android</li><li>• Java</li><li>• ADT plugin</li><li>• Eclipse IDE</li></ul>
-----------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------

---

---

---

---

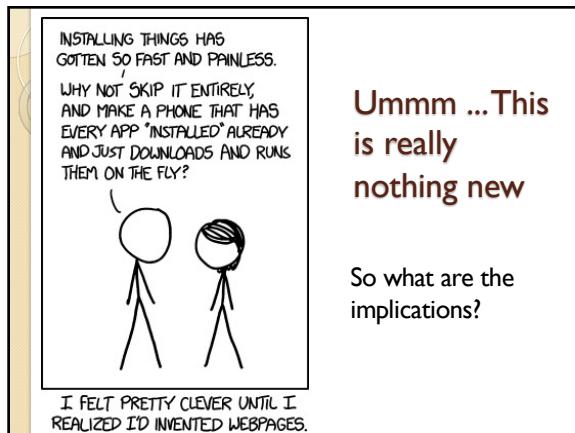
---

---

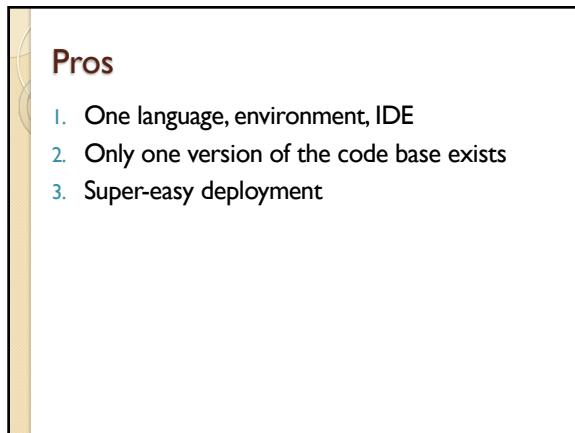
288



289



291



292

---

---

---

---

---

---

---

---

---

---

---

---

---



---

---

---

---

---

---

---

---

---

---

---

---

---



---

---

---

---

---

---

---

---

---

---

---

---

---

## Cons

- It will not be in the AppStore/Play Market/etc.
- It won't have the native look and feel
- Slower
- Cannot access some native resources
  - Accelerometer
  - Proximity sensor
  - GPS (sometimes)
  - Camera (sometimes)
  - Telephone (sometimes)

293

---



---



---



---



---



---



---



---



---

A 10,000 foot view

## Progressive web apps

294

---



---



---



---



---



---



---



---



---

## PWAs requirements that everyone agrees on

- Must be HTTPS
- Runs offline
- User can put an icon on the home screen

295

---



---



---



---



---



---



---



---



---

**Some add these**

- Responsive
- Loads instantly
- Asks the user to add to home screen
- Push notifications
- Runs fast even on slow networks
- Cross-browser
- Page transitions are fast
- Each page gets its own URL

296

---



---



---



---



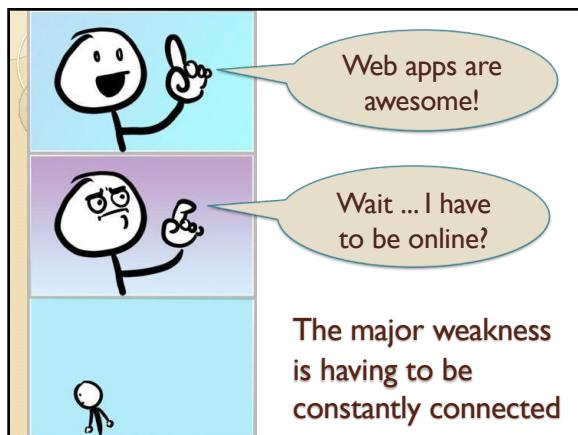
---



---



---



297

---



---



---



---



---



---



---

Email Address	First Name	Last Name	I'm a...	Last Updated	Data Added
aaron@generitech.biz	Aaron	Watson	Designer	Two weeks ago	3/29/2013 10:56AM
jason@generitech.biz	Jason	Beards	Developer	Two weeks ago	3/29/2013 10:56AM
freddie@generitech.biz	Freddie	von Demp	Boss	Two weeks ago	3/29/2013 10:56AM
alex@generitech.biz	Alex	Cohen	Developer	Two weeks ago	3/29/2013 10:56AM
tyrik@generitech.biz	Tyrik	Hobbes	Designer	Two weeks ago	3/29/2013 10:56AM
fed@generitech.biz	Fed	Schneemann	Developer	Two weeks ago	3/29/2013 10:56AM
mardav@generitech.biz	Mardav	Brands	Developer	Two weeks ago	3/29/2013 10:56AM
catalin@generitech.biz	Catalin	Celbi	Designer	Two weeks ago	3/29/2013 10:56AM
gregg@generitech.biz	Gregg	Chernov	Researcher	Two weeks ago	3/29/2013 10:56AM

298

---



---



---



---



---



---



---

## Our strategy

- To create an experience that convinces the user that he is online when he is actually not.
- We'll then re-sync to the database when we're back online.
- A service worker will pull everything offline
- Local Storage can store changes to the data
- onLine/offLine events will tell us when connectivity has changed

299

---



---



---



---



---



---



---



---

## But how do they run it when they can't get to it?

1. Connect to the Internet
2. Download and cache our web app
3. Go offline
4. Run their local version, saving data locally
5. Go back online when possible
6. Synchronize any data with the home base
7. Do steps 1 and 2 only once. Steps 3-6 over and over

300

---



---



---



---



---



---



---



---

## To go offline, you have to have a manifest

```
<!DOCTYPE html>
<html>
<head>
<link rel="manifest" href="/manifest.json">
</head>
<body>
...
</body>
</html>
```

301

---



---



---



---



---



---



---



---



302

A  
manifest  
lists what  
is needed  
locally

---

---

---

---

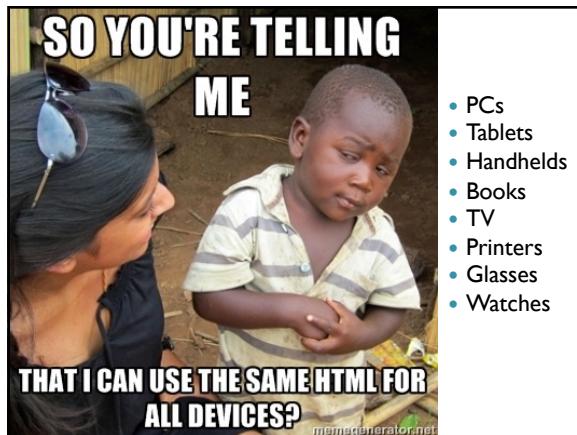
---

---

---



316



317

---

---

---

---

---

---

---



---

---

---

---

---

---

---



---

---

---

---

---

---

---



---

---

---

---

---

---

---



---

---

---

---

---

---

---

CSS3 Media Queries can solve this problem for us to *an extent*

We simply say if you're on a (device name here), make it look like (layout here).

- The buzzword for this is ...

## Responsive Design

318

---



---



---



---



---



---



---



---

Responsive & Adaptive both solve this problem

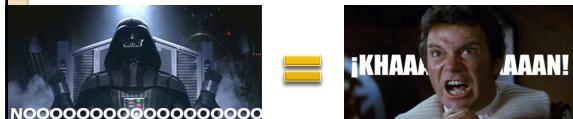
### Responsive Design

- Sections resize with the width
- It is liquid within bands

### Adaptive Design

- Sections stay the same width
- It is static within bands

They're essentially the same



319

---



---



---



---



---



---



---



---

Adaptive is static.  
Responsive is liquid.

**Responsive**



**Adaptive**

320

---



---



---



---



---



---



---



---



321

---

---

---

---

---

---

---

## Two tactics

1. Have a different stylesheet for each view
  - <link ... href="pc.css" />
  - <link ... href="handheld.css" />
  - <link ... href="print.css" />
2. Have one stylesheet with separate sections
  - The sections can have a conditional on them

322

---

---

---

---

---

---

---

## The media attribute is one key

How it might look in our CSS	W3C-recognized types
<pre>@media print {   /* Print layout here */ } @media screen {   /* Screen layout here */ }</pre>	<ul style="list-style-type: none"> <li>• braille</li> <li>• embossed</li> <li>• handheld</li> <li>• print</li> <li>• projection</li> <li>• screen</li> <li>• speech</li> <li>• tty</li> <li>• tv</li> </ul>

323

---

---

---

---

---

---

---



324

---



---



---



---



---



---



---



---

### Combine them with "and" to point to different screen types

- In the <style> or .css file:
 

```
@media screen and (max-width: 960px)
{
 ...
}
```
- In the <head> section
 

```
<link rel="stylesheet" media="screen and
(max-width: 768px)" href="iPad.css" />
```

325

---



---



---



---



---



---



---



---

### What things can you look for?

<ul style="list-style-type: none"> <li>• resolution</li> <li>• orientation</li> <li>• device-aspect-ratio</li> <li>• color   monochrome</li> </ul>	<ul style="list-style-type: none"> <li>• width</li> <li>• max-width</li> <li>• min-width</li> <li>• device-width</li> <li>• max-device-width</li> <li>• min-device-width</li> <li>• all the above for height, too</li> </ul>
----------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

326

---



---



---



---



---



---



---



---

## Three approaches to bands

### Big to Small

```
/* Large display - unqualified */
@media (max-width: 992px) {/* Regular display */}
@media (max-width: 768px) {/* Tablet */}
@media (max-width: 480px) {/* Phone */}
```

### Small to Big

```
/* Phone - unqualified */
@media (min-width: 768px) {/* Tablet */}
@media (min-width: 992px) {/* Regular display */}
@media (min-width: 1200px) {/* Large display */}
```

### Unambiguous bands

```
@media (max-width: 767px) {/* Phone*/}
@media (min-width: 768px)
 and (max-width: 991px) {/* Tablet */}
@media (min-width: 992px)
 and (max-width: 1199px) /* Regular display */
@media (min-width: 1200px) {/* Large display */}
```

327

## tl;dr

- PWAs have tons of requirements but they allow the use of our web apps offline.
- Media queries allow us to apply different styles to different media and different screen sizes
- We can use these to have responsive web designs that flex when browsed on different devices

334

## Modern CSS Formatting

How to make the web look great using modern techniques

338

**tl;dr**

- CSS styles give the site its look and feel by setting colors, fonts, sizes, layouts, spacing and so much more
- Cool tricks are being added all the time like image filters, data uris, gradients, shadows, rounded corners and more

340

---



---



---



---



---



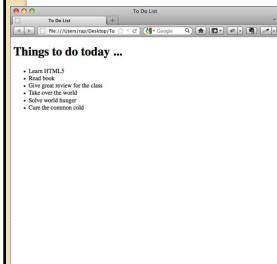
---



---

**We can make tons of adjustments to look and feel with CSS alone**

We can take this:



And make it look like this:



341

---



---



---



---



---



---



---

**Styling Text**

342

---



---



---



---



---



---



---

## What properties can be set?

- Colors
- Fonts
- Decoration
- Locations
- So many more!!

---



---



---



---



---



---



---

343

## Colors

- color: name|hex  
value|rgb(R,G,B)|rgba(R,G,B,A)|hsl(H,S,L)|hsla(H,S,L,A)

red  
green  
blue  
white  
black  
purple  
orange  
...

- A value will be three hex numbers 00-FF
- The first is the red portion
- The second is green
- The third is blue
- 16,777,216 combinations

---



---



---



---



---



---



---

344

## font-style

- font-style:  
normal|italic|oblique  
font-style: italic;




---



---



---



---



---



---



---

346

## font-weight

- font-weight: normal|bold|bolder|lighter|100-900
- a number
  - They go thin to thick
  - 400 is normal
  - 700 is bold
- font-weight: 700;
- bolder means thicker than its parent
- lighter is the opposite of bolder

347

---



---



---



---



---



---



---



---



---



---

## font-size

- font-size: relative size|explicit size;

normal	large	Xpx
xx-small	x-large	Xpt
x-small	xx-large	Xem
small	smaller	X%
medium	larger	

- font-size: relative size|explicit size;  
font-size: 2em;

348

---



---



---



---



---



---



---



---



---



---

## font-family

- font-family: font1 [, font2 [, ...]]
  - Traverses the list until it finds a font it can use.
  - Always put a default at the end
    - serif
    - sans-serif
- ```
body {
  font-family: Verdana, Arial, "Comic Sans", sans-serif;
}
```

349

Web fonts allow you to expand beyond the installed fonts

- When your site needs a font, it pulls it off the Internet
- Your site
- Someone else's (Google, for instance)

350

Here's an example

```
@font-face {
  font-family: funkyfont;
  src: url(somesite.com/funkyfont.ttf);
}
.funky {
  font-family: funkyfont;
}
```

351

text-align: left|right|center|justify

Flush left / Ragged right

Left-aligned text is the most legible option for web pages. It's less formal and more inviting than the fully justified type. And it's much easier to maintain well formed text blocks without odd spaces in the middle of the text.

Centered

Centered alignment is a weak choice for long bodies of text. The best possible scenario for a center-alignment is when there is very little on a page, such as titles or small pieces of information that require special attention.

Flush right / Ragged left

Right-aligned text can be used in the left column of a page or table to show a closer relationship between the elements in adjacent columns and meant to sit closer to the right edge of the screen.

Justified

Justified text is very readable when set properly. It allows for a higher word density. Be aware of the white space in the middle of the text, which creates a visual problem that requires adjustment of the lines of types.

352

text-decoration

- `text-decoration: none|underline|overline|line-through`
- To do multiples, list them both with no comma
`text-decoration: underline;`



353

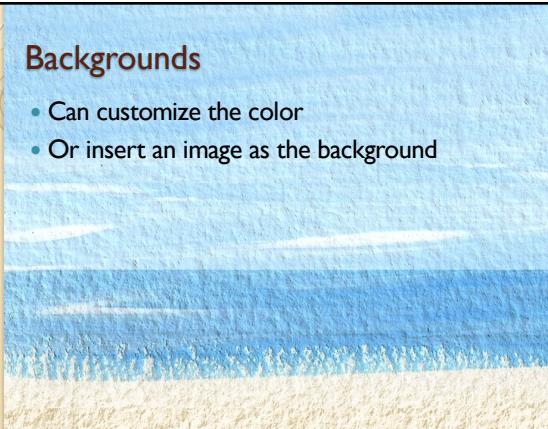
Who doesn't like tips?

css Tips and Tricks!

355

Backgrounds

- Can customize the color
- Or insert an image as the background



356

When to use vs background?

| | |
|---|--|
| <code></code> | <code>#someDiv {
background: url("foo.jpg");
}</code> |
| Makes room for the size of the image | Lets the <div> size itself |
| Changing the size of the image changes the layout | Changing the size of the background has no effect on the layout |
| Seen by ally | Not seen by ally |
| So, use this when you want the img to drive the layout of the page or when it would be important to ally. | So use this when you want the image to be decoration for a section that should size itself and is unimportant to ally. |

357

How do I resize the background image?



```
background-repeat: no-repeat;  
background-size: cover;  
/* or contain */  
/* or "100px 125px" */  
background-position: 10px 20px;  
/* horizontal vertical;  
Can use...  
- "top"  
- "bottom"  
- "center"  
- "left"  
- "right"
```

358

| | | |
|---------------------------------|--|---|
| Top to bottom | <code>background: linear-gradient(to bottom, white, red);</code> |  |
| Left to right | <code>background: linear-gradient(to right, red, white);</code> |  |
| Bottom right to top left | <code>background: linear-gradient(45deg, red, white);</code> |  |

Color gradients look cool!

359

Data uris may speed up your images

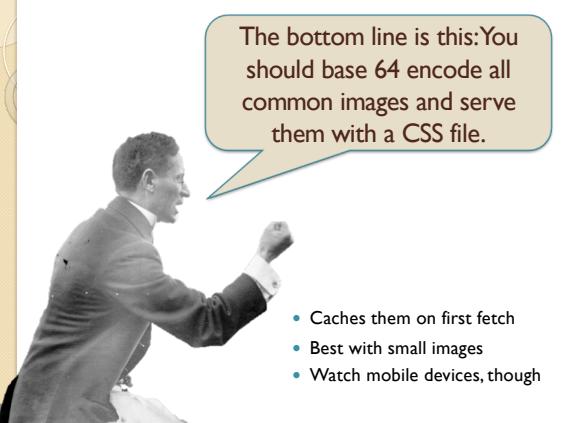


```
.twitterLogo {
  width: 20px;
  height: 20px;
  background-image: url("data:image/png;base64,iVBORw0KGgo...kSuQmCC");
}
```

| | |
|--|--|
| Pros | Cons |
| <ul style="list-style-type: none"> • Fewer HTTP requests! | <ul style="list-style-type: none"> • Size is increased by 1/3 • Browser has to process the image |

So what should we do?

362



The bottom line is this: You should base 64 encode all common images and serve them with a CSS file.

- Caches them on first fetch
- Best with small images
- Watch mobile devices, though

363

CSS Filter property



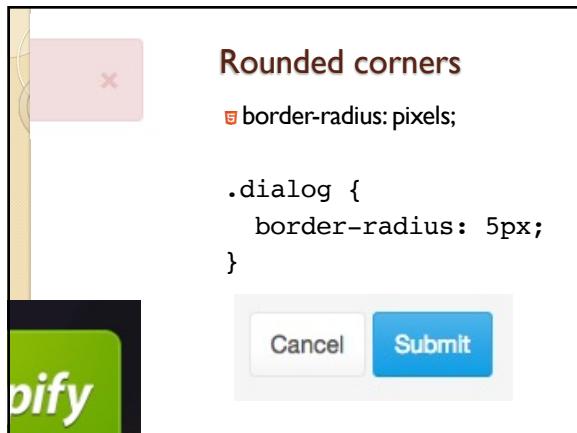
Applies filters to images!

```
.someImg {
  filter:
    blur(25px) /* Bigger numbers => more blurry */
    brightness(10%)
    contrast(10%)
    drop-shadow()
    grayscale(50%) /* 100% = totally gray */
    hue-rotate(180) /* degrees rotation */
    invert(100%) /* 100% = totally inverted */
    opacity(50%) /* May be faster */
    saturate()
    sepia(0.8) /* 1 = totally sepia */
}
```

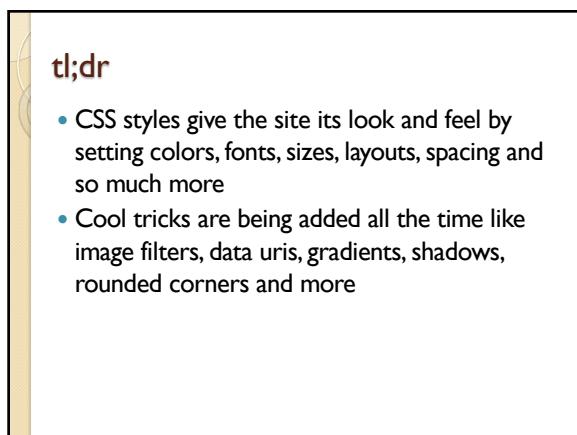
364



366



367



369



Advanced CSS Selectors

How to point to almost anything

373



tl;dr

- CSS selectors allow us great flexibility in pointing to elements on a page
- We can point to elements by id, class, and type but they go much deeper. You can select elements by relationship, position, attributes, state, and more.
- Yes, they're tough to learn but we can use them for styles and tons of libraries like Angular and jQuery

375



Remember the basic style syntax

```
selector {  
  property: value;  
  [property: value ...]  
}
```

376

With that selector-thingy, you can point to just about anything

- A single element
- A group of elements
- All elements of a type
- All descendants of an element
- Sibling elements
- Just the nth child
- The active element
- One we're hovering over
- ...

377

So why should we learn this stuff?

- For ninja-like precision with ...
- CSS and styles
 - `document.querySelector()`
 - `document.querySelectorAll()`
 - Angular
 - jQuery
 - Other tools/Frameworks

378



380

Remember your basic
selectors?

381

Element selectors point to all elements
of a certain type

```
body {  
    font-family: arial, sans-  
    serif;  
}  
p {  
    padding: .5em;  
}
```



382

Class selectors allow you to group
elements any way you see fit

- `.className {color: red;}`
- To apply a class in HTML
`<p class="fancy">`
- To apply two or more classes
`<p class="fancy important">`
- This will combine the characteristics of both
classes

383

ID selectors point to exactly one thing



```
n {  
 7;
```

384



Combining selectors

385

Concatenating = all requirements

```
tr.fancy {  
  font-family: cursive;  
}  
p.headline {  
  font-size: 4em;  
  font-family: courier serif;  
}  
.touched.invalid {  
  color: red;  
  border: 2px solid red;  
}
```

386

A comma between means that you're applying a style to two or more selectors

```
p.heavy, #firstName {
  font-weight: bold;
}
```

All <p>s with a class of heavy



or

The first thing with an id of "firstName"

387

Grouping selectors can add to the organization

```
.headlines{
  font-family:arial;
  color:black;
  background:yellow;
  font-size:14pt;
}
.sublines {
  font-family:arial;
  color:black;
  background:yellow;
  font-size:12pt;
}
.infotext {
  font-family:arial;
  color:black;
  background:yellow;
  font-size:10pt;
}

.headlines, .sublines, .infotext {
  font-family:arial;
  color:black;
  background:yellow;
}
.headlines {font-size:14pt;}
.sublines {font-size:12pt;}
.infotext {font-size:10pt;}
```

391

Selecting by DOM position

 Relationship selectors

393

The DOM also points to relationships between elements

- Descendants
- Parent
- Child
- Siblings
- Ancestors



394

Descendant selectors

ancestor-sel descendant-sel

- With a space between them
 - This will point to descendants of ancestor-sel at ANY level below
- ```
div#sidePanel li {
 color: blue;
}
```

---

---

---

---

---

---

---

395

## Child selectors

parent > child

- Selects only **direct** children

```
div#sidePanel > p {
 font-style: italic;
}
```




---

---

---

---

---

---

---

396

## Sibling selectors

`sel1 ~ sel2`

- Applies to all sel2 elements that are ...
  - at the same level as a sel1 element and ...
  - after the sel1 element

```
li#special ~ li {
 background-color: red;
}
```

397

---



---



---



---



---



---



---



---

## Adjacent sibling selectors

`sel1 + sel2`

- Same as before, but only if sel2 is immediately after sel1

```
h2 + h3 {
 margin: -1em;
```

398

---



---



---



---



---



---



---



---

## Attribute selectors

400

---



---



---



---



---



---



---



---

## Remember XML attributes?

- Attributes are usually descriptors of an element.

```
<p class="fancy">

<button value="Click me">
```

- You can select elements by HTML attribute

401

---

---

---

---

---

---

---

## Attribute exists

- sel[attr]
  - The attribute is *present*
- ```
img[alt]  
a[href]
```

402

Attribute is exactly ...

- sel[attr="value"]
 - The attribute is set to "value"
- ```
img[src="help.gif"]
```

403

---

---

---

---

---

---

---



### Attribute starts with ...

- sel[attr^="startsWith"]
- The attribute starts with

```
img[alt^="picture of"]
```

---

---

---

---

---

---

---

404



### Attribute ends with ...

- sel[attr\$="endsWith"]
- The attribute ends with

```
a[href$="google.com"]
```

---

---

---

---

---

---

---

405



### Attribute contains ...

- sel[attr\*="contains"]
- The attribute contains

```
a[href*="google.com"]
```

---

---

---

---

---

---

---

406

**Pseudo-classes**

407

---

---

---

---

---

---

**Pseudo-classes are like classes you get with no coding on your part!**

- "pseudo" = "resembling but not genuine"
  - Like pseudoscience or pseudonym or pseudocode
- "classes" = CSS classes
- So you can select based on things outside the DOM.
- Always has a colon
- Occasionally called 'CSS states' because they can change through user interaction

408

---

---

---

---

---

---

**Pseudo-classes for links**

- `:link` – The anchor is a link
- `:visited` – We've visited that link recently

409

---

---

---

---

---

---

## User-action pseudo-classes for links

- `:hover`
  - We're hovering over the link
- `:active`
  - We're clicking on that link right now
- `:focus`
  - The one the user clicks on or tabs into

410

---



---



---



---



---



---



---



---



---



---

## Form-related pseudo-classes

- `:enabled & :disabled`
- `:valid & :invalid`
- `:required & :optional`
- `:read-only & :read-write`
- `:in-range & :out-of-range`
  - Satisfies or violates the min/max attribute
- `:checked & :indeterminate`

411

---



---



---



---



---



---



---



---



---



---

## Structural pseudo-classes

- `:first-child`
  - A child, but just the first one encountered
- `:last-child`
  - A child, but just the last one encountered
- `:only-child`
  - Anything with no siblings
- `:only-of-type`
  - Anything that has no siblings of the same type
- `:nth-of-type(XXX)`
- `:first-of-type() & :last-of-type()`
- `:only-of-type` – Will only style if it is the only one of its type in the parent.

412

---



---



---



---



---



---



---



---



---



---

## nth-child(something)

- The *something* can be
- "even"
- "odd"
- X (a number)
- Xn
- Xn+Y
- Xn-Y



There's also a nth-last-child() that counts backwards from the end

413

---

---

---

---

---

---

---

## Negation pseudo-class

- :not(simple selector)
- Matches everything that is NOT the selector

```
p:not(:first-child) {
 /* All paragraphs that are not the first child */
}

a:not(:visited):not(:hover) {
 /* Links neither visited nor being hovered */
}
```

A *simple selector* is ...

- type
- class
- ID
- another pseudo-class

414

---

---

---

---

---

---

---

## Pseudo-Elements

415

---

---

---

---

---

---

---

## Pseudo-elements

- ::first-line
  - Only the first line of a paragraph
- ::first-letter
  - Just the first letter

417

---



---



---



---



---



---



---



---

## ::before and ::after insert content inside an element

- Example

```
p.hasLogo::before {
 content: url(logo.jpg);
}
```

418

---



---



---



---



---



---



---



---

## tl;dr

- CSS selectors allow us great flexibility in pointing to elements on a page
- We can point to elements by id, class, and type but they go much deeper. You can select elements by relationship, position, attributes, state, and more.
- Yes, they're tough to learn but we can use them for styles and tons of libraries like Angular and jQuery

419

---



---



---



---



---



---



---



---



## CSS3 Transforms and Transitions

How to create amazing effects and motion without JavaScript!

---



---



---



---



---



---

421



### tl;dr

- Transforms can make our sites come alive with translate, scale, rotate, and more
- The transition features allow us to create smooth transforms with control over duration, initial delay, and the easing function
- We no longer need JavaScript. It can all be done with pure CSS
- We can also combine transitions to make it really look good

---



---



---



---



---



---

423



### Old-school transitions were done with JavaScript and dynamic HTML

```
<script language="javascript">
var theDiv =
 document.getElementById("theDiv");
var y = 5; //StartLocation
var destY = 300; //EndLocation
function moveImage() {
 if(y < dest_y) y = y + 10;
 theDiv.style.top = y +'px';
 if (y + 10 < dest_y) {
 window.setTimeout('moveImage()',100);
 }
}</script>
```

---



---



---



---



---



---

424

**CSS transforms can now alter DOM elements with styles**

- We can change
  - location (translate)
  - angle (rotate)
  - size (scale)
  - distortion (skew)



425

---



---



---



---



---



---



---



---

**Scaling can be done through brute force**

```
#theDiv {
 height: 10px;
 width: 10px;
}
#theDiv:hover {
 height: 25px;
 width: 20px;
}
```

426

---



---



---



---



---



---



---



---

**Scaling can be done by using transform**

```
#theDiv {
 height: 10px;
 width: 10px;
}
#theDiv:hover {
 transform: scale(2.5, 2);
```

427

---



---



---



---



---



---



---



---

```
div.move { width: 50px; height: 50px; padding: 10px; margin-left: 0px; }
div.move:hover { margin-left: 40px; margin-top: 40px; }
```

**Location can be done by brute force ...**



429

```
div.move { width: 50px; height: 50px; padding: 10px; margin-left: 0px; }
div.move:hover { transform: translate(40px 40px); }
```

**... or by using transform**

430

## Rotation

```
img.verticalLogo {
 transform: rotate(-90deg);
}
• Angle in ...
◦ deg
◦ rad
◦ grad
◦ turn
```

432

---

---

---

---

---

---

---



---

---

---

---

---

---

---



---

---

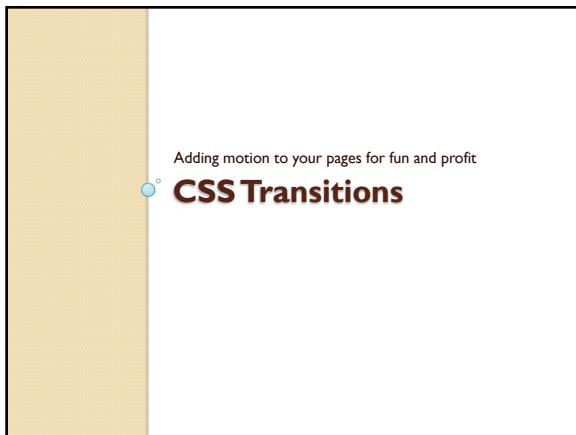
---

---

---

---

---



438

---

---

---

---

---

---

A slide with a yellow vertical bar on the left. The main content area has a title 'Intro to transitions' and a bulleted list: 'The transforms section showed us how to change HTML elements' and 'Now let's see how to change them gradually'.

439

---

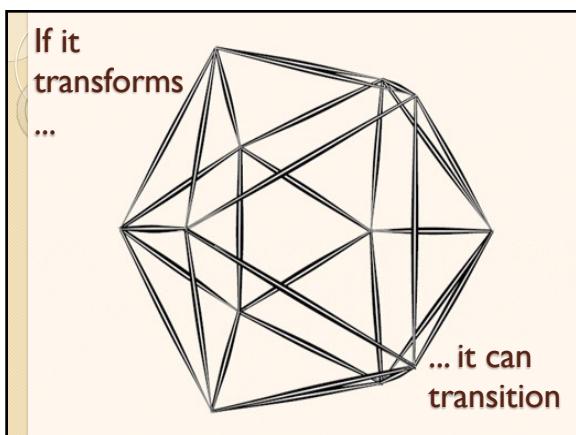
---

---

---

---

---



440

---

---

---

---

---

---

On the element you want to have  
animate, add some CSS

```
cssSelector {
 transition: all 1s ease-in-out;
}
```

441

---



---



---



---



---



---



---



---

Full syntax of transitions

```
foo {
 transition-property: <property>;
 transition-duration: <time>;
 transition-timing-function: <functionName>;
 transition-delay: <time>;
}

...or more succinctly ...
foo {
 transition: <property> <duration>
 <timing-function> <delay>;
}
```

444

---



---



---



---



---



---



---



---

The property is the thing you want  
transitioned

eg ...

- background-color
- height
- width
- top
- left
- transform

... or ...

- **all**

445

---



---



---



---



---



---



---



---

The duration is how long we should take to transition from the first state to the last

- In seconds or milliseconds
- 5s
- 5000ms

446

---

---

---

---

---

---

The timing function tells how the transition accelerates over time

- ease
- linear
- ease-in
- ease-out
- ease-in-out
- cubic-bezier(x1,y1,x2,y2)

447

---

---

---

---

---

---

The transition delay tells how long to wait before beginning

- in Seconds (s) or milliseconds (ms)

449

---

---

---

---

---

---

 Put them all together like so ...

```
#theDiv {
 width: 10px;
 transition-property: width;
 transition-duration: 2s;
 transition-timing-function: ease-in-out;
 transition-delay: 1s;
}
... or ...
#theDiv {
 width: 10px;
 transition: width 2s ease-in-out 1s;
}
```

450

---



---



---



---



---



---



---



---

 We can combine transitions

```
#theDiv {
 transition-property: top, left;
 transition-duration: 3s, 1s;
 transition-delay: 0s, 3s;
}
```

451

---



---



---



---



---



---



---



---

 ... but this is not okay

```
#theDiv:hover {
 transition: top 3s;
 transition: left 1s;
}
• Why?
• Because the second overwrites the first.
```

452

---



---



---



---



---



---



---



---



### tl;dr

- Transforms can make our sites come alive with translate, scale, rotate, and more
  - The transition features allow us to create smooth transforms with control over duration, initial delay, and the easing function
  - We no longer need JavaScript. It can all be done with pure CSS
  - We can also combine transitions to make it really look good
- 
- 
- 
- 
- 
- 

456



### Deep Dive into Tables

Presenting data in rows and columns

---

---

---

---

---

---

458



### tl;dr

- HTML tables are for data, not for layout
  - They can have headers and footers
  - Cells can span rows and columns
  - We can make tables look fantastic by styling them through CSS
- 
- 
- 
- 
- 
- 

460

## Sometimes data is best presented in tables

- Rows
- Columns
- Cells
- With headers and footers

	January	February	March	Total
Andy Bernard	\$9,963.49	\$5,976.20	\$7,920.45	\$23,860.14
Pam Halpert	\$8,258.87	\$6,188.07	\$9,365.33	\$23,812.27
Stanley Hudson	\$5,587.17	\$9,493.05	\$5,911.82	\$20,992.04
Phyllis Vance	\$7,908.27	\$8,352.94	\$6,962.47	\$23,223.68
Jim Halpert	\$9,028.92	\$9,621.17	\$9,072.50	\$27,722.59
Dwight Schrute	\$5,768.76	\$6,072.45	\$8,563.50	\$20,404.71
	\$46,515.48	\$45,703.88	\$47,796.08	\$140,015.43

461

IF LAYING OUT WITH TABLES IS SO WRONG



Tables  
are for  
data  
**ONLY**

WHY DOES MY PAGE LOOK SO  
RIGHT?

memegenerator.net

Tables are simple to create, but verbose

```
<table> ... </table>
<tbody> ... </tbody>
<tr> ... </tr>
<td> ... </td>
```



463

## A simple table has just rows and columns

```
<table>
<tbody>
<tr>
<td>Andy
Bernard</td><td>$9,963.49</td><td>$5,976.20</td><td>$7,920.45</td>
$23,860.14</td>
</tr>
<tr>
<td>Pam
Halpert</td><td>$8,258.87</td><td>$6,188.07</td><td>$9,365.33</td>
$23,812.27</td>
</tr>
<tr>
<td>Stanley
Hudson</td><td>$5,587.17</td><td>$9,493.05</td><td>$5,911.82</td>
$20,992.04</td>
</tr>
...
</tbody>
</table>
```

464

## But to be proper, we should have header, footer & body

```
<thead>
<tr>
<th></th><th>January</th><th>February</th><th>March</th>
<th>Total</th>
</tr>
</thead>
<tfoot>
<tr><td>$46,515.48</td><td>$45,703.88</td>
<td>$47,796.08</td><td>$140,015.43</td></tr>
</tfoot>
<tbody>
<tr><td>Andy
Bernard</td><td>$9,963.49</td><td>$5,976.20</td><td>
$7,920.45</td><td>$23,860.14</td></tr>
...
</tbody>
```

465

## We can have data that spans across two or more rows or columns

```
<tr>
<th colspan="5">First Quarter Sales</th>
</tr>
```



467



468

---

---

---

---

---

---

### Applying overall styles

```
<link rel="stylesheet" href="site.css" />
table{
 font-family: Arial, Sans-Serif;
 font-size: 12px;
 background: #fff;
 margin: 45px;
 width: 480px;
}
th{
 font-size: 14px;
 color: #039;
 padding: 10px 8px;
 border-bottom: 2px solid;
}
td{
 color: #669;
 padding: 9px 8px 0px 8px;
}

```

	January	February	March	Total
Andy Bernard	\$9,963.49	\$5,976.20	\$7,920.45	\$23,860.14
Pam Halpert	\$8,258.87	\$6,188.07	\$9,365.33	\$23,812.27
Stanley Hudson	\$5,587.17	\$9,493.05	\$5,911.82	\$20,992.04
Phyllis Vance	\$7,908.27	\$8,352.94	\$6,962.47	\$23,223.68
Jim Halpert	\$9,028.92	\$9,621.17	\$9,072.50	\$27,722.59
Dwight Schrute	\$5,768.76	\$6,072.45	\$8,563.50	\$20,404.71
	\$46,515.48	\$45,703.88	\$47,796.08	\$140,015.43

469

---

---

---

---

---

---

### Alternating rows

```
tr:nth-child(odd) {
 background: #eee;
}

```

	January	February	March	Total
Andy Bernard	\$9,963.49	\$5,976.20	\$7,920.45	\$23,860.14
Pam Halpert	\$8,258.87	\$6,188.07	\$9,365.33	\$23,812.27
Stanley Hudson	\$5,587.17	\$9,493.05	\$5,911.82	\$20,992.04
Phyllis Vance	\$7,908.27	\$8,352.94	\$6,962.47	\$23,223.68
Jim Halpert	\$9,028.92	\$9,621.17	\$9,072.50	\$27,722.59
Dwight Schrute	\$5,768.76	\$6,072.45	\$8,563.50	\$20,404.71
	\$46,515.48	\$45,703.88	\$47,796.08	\$140,015.43

471

---

---

---

---

---

---

## Hovering over a row: tr:hover

```
tbody tr:hover td
{
 color: #339;
 background: #d0dafd;
}
```

First Quarter Sales

	January	February	March	Total
Andy Bernard	\$9,963.49	\$5,976.20	\$7,920.45	\$23,860.14
Pam Halpert	\$8,258.87	\$6,188.07	\$9,365.33	\$23,812.27
Stanley Hudson	\$5,587.17	\$9,493.05	\$5,911.82	\$20,992.04
Phyllis Vance	\$7,908.27	\$8,352.94	\$6,962.47	\$23,223.68
Jim Halpert	\$9,028.92	\$9,621.17	\$9,072.50	\$27,722.59
Dwight Schrute	\$5,768.76	\$6,072.45	\$8,563.50	\$20,404.71
	\$46,515.48	\$45,703.88	\$47,796.08	\$140,015.45

472

## tl;dr

- HTML tables are for data, not for layout
- They can have headers and footers
- Cells can span rows and columns
- We can make tables look fantastic by styling them through CSS

474

## • Best Practices with Forms

... because a web session is a dialogue, not a monologue

476

**tl;dr**

- Forms are used to collect data from the user and then send them to the server for processing
- There are a few fields like textarea, select, datalist
- And there are lots of <input> fields with types like text, search, number, range, and email

478

---

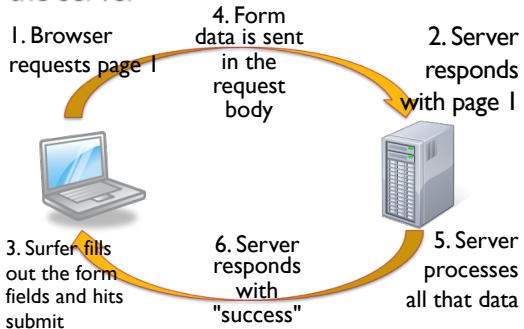
---

---

---

---

---

**Forms are the main way to interact with the server**

479

---

---

---

---

---

---

**HTTP Methods**

GET
HEAD
POST
PUT
PATCH
DELETE
TRACE
CONNECT
OPTIONS

480

---

---

---

---

---

---

The data may look like this...

```
address=2636+Harvard+Road&city=Boston&comp
anyName=Agile+Gadgets&email=tlewous@gmail.
com&firstName=Trale&lastName=Lewous"
```

Or if prettied up ...

```
{
 "address": "2636 Harvard Road"
 "city": "Boston"
 "companyName": "Agile Gadgets"
 "email": "tlewous@gmail.com"
 "firstName": "Trale"
 "lastName": "Lewous"
}
```

481

The <form> tag

```
<form action="formAction.jsp" method="post">
 What's your name?
 <input name="firstName" />

 <input type="submit" />
</form>
```

482

Forms  
post their  
data in the  
**POST**  
request's  
payload



483

---



---



---



---



---



---



---



---



---



---



---



---



---



---



---



---



---



---



---



---



---

## What goes inside the form?

- **textarea**
  - for multiline text
- **select**
  - for dropdown lists and listboxes
- **input**
  - for text fields
  - for checkboxes
  - for radio buttons
  - for file uploads
  - for other things

485

---

---

---

---

---

---

---

---

## textarea is for multiline text in forms

<textarea>  
Digg is a social news website. Prior to Digg v4, its cornerstone function consisted of letting people vote stories up or down, called digging and burying, respectively. Digg's popularity prompted the creation of copycat social networking sites with story submission and voting systems

Digg is a social news website. Prior to Digg v4, its cornerstone function consisted of letting people vote stories up or down, called digging and burying, respectively. Digg's popularity prompted the creation of copycat social

487

---

---

---

---

---

---

---

---

## select can be listboxes or dropdowns

David at the Dentist  
Chocolate Rain  
Double Rainbow  
Numa Numa

David at the Dentist :  
David at the Dentist  
Chocolate Rain  
Double Rainbow  
Numa Numa

```
<select multiple="multiple">
<option>David at the Dentist</option>
<option>Chocolate Rain</option>
<option>Double Rainbow</option>
<option>Numa Numa</option>
</select>
```

```
<select>
<option>David at the Dentist</option>
<option>Chocolate Rain</option>
<option>Double Rainbow</option>
<option>Numa Numa</option>
</select>
```

488

---

---

---

---

---

---

---

---

## inputs are for everything else

```
<input type="_____ " name="whatever" />

• text
• password
• checkbox
• radio
• file
• submit
• button
• hidden
```

|            |            |
|------------|------------|
| • text     | ✉ email    |
| • password | ✉ url      |
| • checkbox | ✉ number   |
| • radio    | ✉ range    |
| • file     | ✉ date     |
| • submit   | ✉ datetime |
| • button   | ✉ month    |
| • hidden   | ✉ week     |
|            | ✉ time     |
|            | ✉ search   |
|            | ✉ color    |

493

---

---

---

---

---

---

---

---

---

---

## The simple input types

- text – for text (duh)
- password – text, but the characters are not typed back to the screen
- checkbox – has a boolean selected value
- radio – a radio button – like a checkbox except that only one can be selected at a time
- submit – the button that submits the form

---

---

---

---

---

---

---

---

---

495

## Radio buttons

- To group them, give them the same name attribute.

```
<p>Your gender:</p>
<input type="radio" name="gender"
 id="m" value="male" />
<label for="m">Male</label>
<input type="radio" name="gender"
 id="f" value="female" />
<label for="f">Female</label>
```

---

---

---

---

---

---

---

---

---

496

### ⌚ **input type="email"**

- A text field, but only accepts values that look like email addresses

E-mail (email)

abc

abc is not a legal e-mail address

datetime) : UTC

500

---

---

---

---

---

---

### The keyboard changes with a type set



501

---

---

---

---

---

---

### ⌚ **input type="number"**

- A number

```
<input type="number"
 min="0"
 max="10"
 step="0.5"
 value="6" />
```

504

---

---

---

---

---

---

 **5 input type="range"**

- Also a number

```
<input type="range"
 min="0"
 max="10"
 step="2"
 value="6" />
```

Range (range)



505

---



---



---



---



---



---



---

 **6 <input type="date" />**

Date (date)



| Week | Mon | Tue | Wed | Thu | Fri | Sat | Sun |
|------|-----|-----|-----|-----|-----|-----|-----|
| 13   | 29  | 30  | 31  | 1   | 2   | 3   | 4   |
| 14   | 5   | 6   | 7   | 8   | 9   | 10  |     |
| 15   | 12  | 13  | 14  | 15  | 16  | 17  | 18  |
| 16   | 19  | 20  | 21  | 22  | 23  | 24  | 25  |
| 17   | 26  | 27  | 28  | 29  | 30  | 1   | 2   |
| 18   | 3   | 4   | 5   | 6   | 7   | 8   | 9   |

Today      None

507

---



---



---



---



---



---



---

 **7 Form attributes**

512

---



---



---



---



---



---



---

## HTML5 attributes

- ⌚ autofocus
- ⌚ min, max, and step
- ⌚ placeholder
- ⌚ pattern
- ⌚ required
- ⌚ multiple
- ⌚ datalist

513

---



---



---



---



---



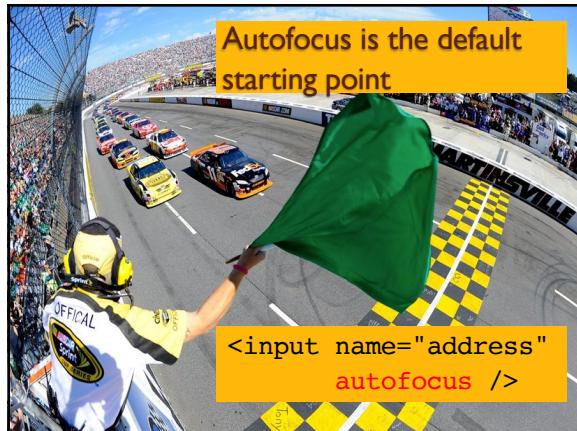
---



---



---



514

---



---



---



---



---



---



---



---

## ⌚ Placeholder puts ghost text in the textbox

```
<input type='text' name='firstName'
placeholder='First Name' />
```

|               |                                                   |
|---------------|---------------------------------------------------|
| First Name    | <input type="text" value="First Name"/>           |
| Last Name     | <input type="text" value="Last Name"/>            |
| Email address | <input type="text" value="anything@example.com"/> |

515

---



---



---



---



---



---



---

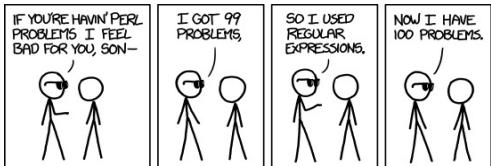


---

### 5 Built-in validation happens when you specify a pattern

- a regular expression

```
<input name='creditCard'
pattern='(\d{4}[-]?)?(\d{3}[-]?)?\d{4}'
title='Enter a credit card (XXXX-XXXX-
XXXX-XXXX)' />
```



516

---

---

---

---

---

---

---

---

### 5 Requiring values

- Add "required" to any form element

```
<input name="email" type="email" required />
```

---

---

---

---

---

---

---

---

517

### tl;dr

- Forms are used to collect data from the user and then send them to the server for processing
- There are a few fields like textarea, select, datalist
- And there are lots of <input> fields with types like text, search, number, range, and email

---

---

---

---

---

---

---

---

523