How All This Actually Works

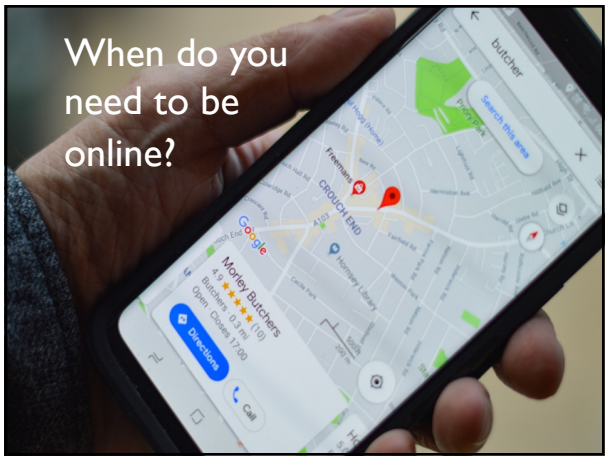
The secrets behind the web

1

tl;dr

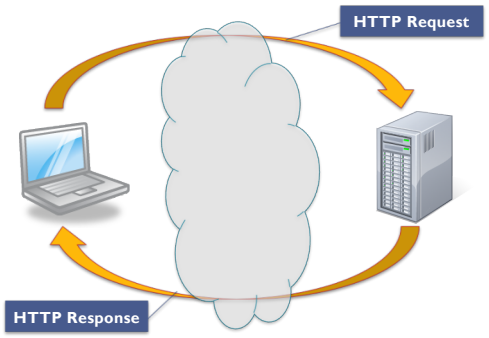
- Browsers run thin-client ... sort of
- They use HTTP and HTML, a static DSL
- They don't conform to a standard but the W3C/WHATWG makes one anyway
- The specs are not a great source of help. MDN and caniuse.com are excellent

2

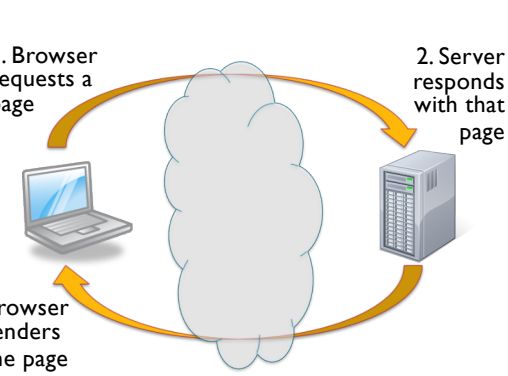


4

The web uses a thin client architecture



5



6

… because HTML is a markup language

Introducing *ML

7

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

9

SGML is the original standard

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

10

HTML is the language of the web

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

11

XML

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

It needs to be
- Well-formed
- Valid

12

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

13

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>!
```

14

Different browsers lead to different renderings

15

How do the browser manufacturers know what to draw?

World Wide Web Consortium

Web Hypertext App Tech Working Group

W3C and WHATWG publish standards

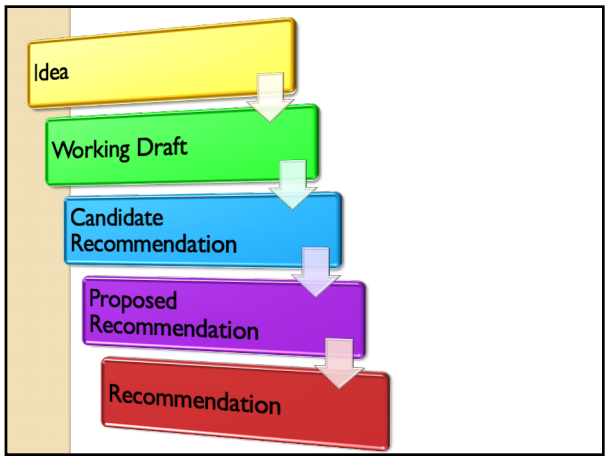
No one is required to comply

17

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

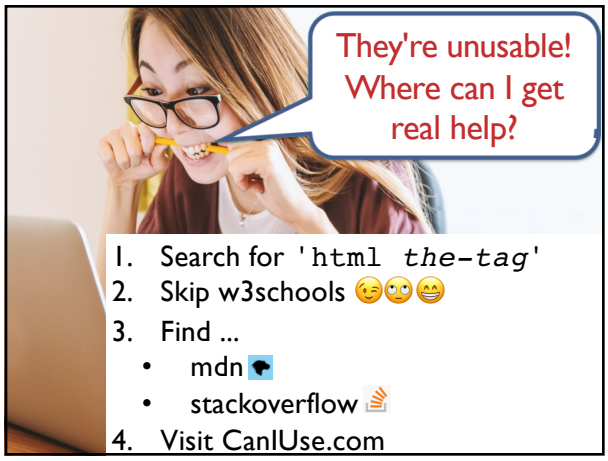
18


Idea
Working Draft
Candidate Recommendation
Proposed Recommendation
Recommendation

19


These specs are so hard to find!
https://html.spec.whatwg.org/
https://www.w3.org/TR/

20


They're unusable! Where can I get real help?
1. Search for 'html the-tag'
2. Skip w3schools 😉🙄😄
3. Find ...
• mdn
• stackoverflow
4. Visit CanIUse.com

21

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/WHATWG makes one anyway
- The specs are not a great source of help. MDN and caniuse.com are excellent

28

Debugging your HTML and CSS

29

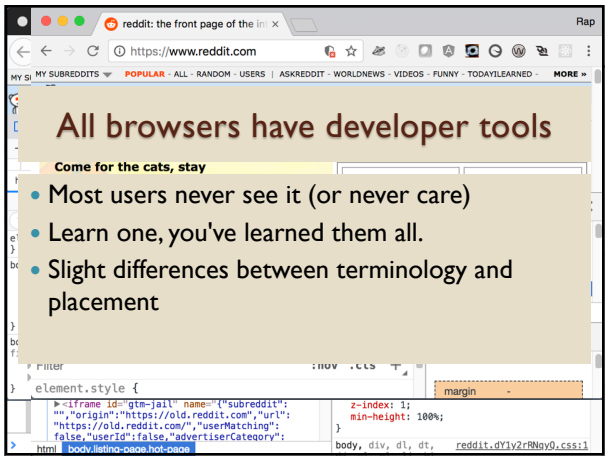
tl;dr

- 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

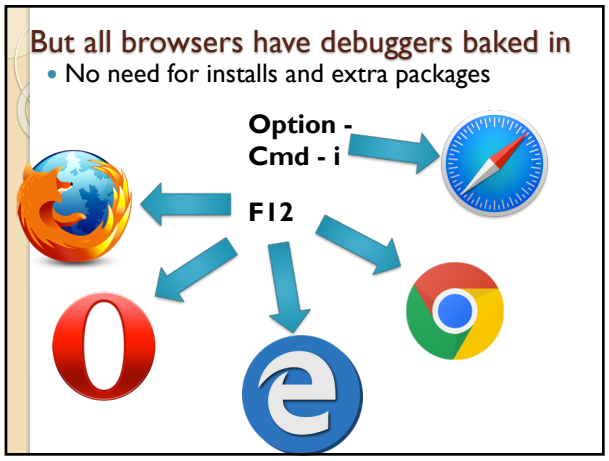
31


HTML and CSS run in the browser so they must be debugged in the browser

32

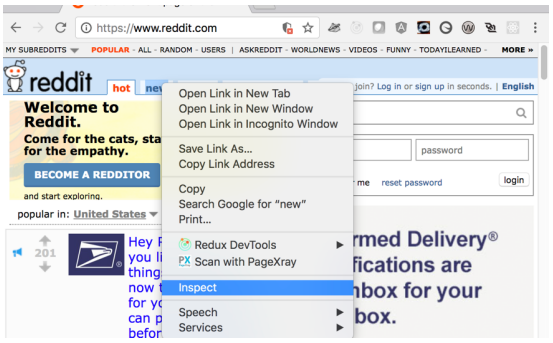

All browsers have developer tools
Come for the cats, stay
• Most users never see it (or never care)
• Learn one, you've learned them all.
• Slight differences between terminology and placement

33


But all browsers have debuggers baked in
• No need for installs and extra packages
Option - Cmd - i
F12

34

... Or you can choose 'inspect'



35

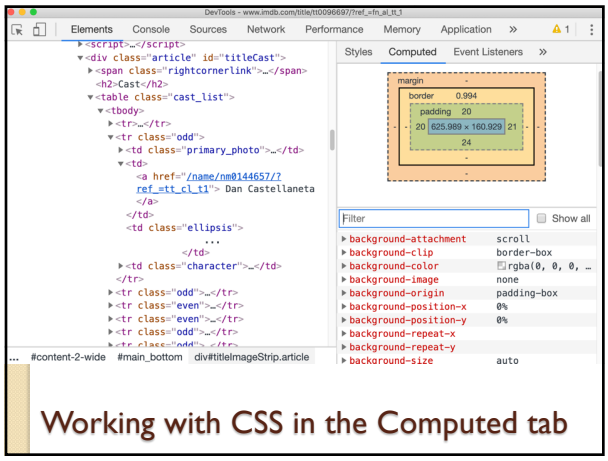
You can actually make changes to the CSS and HTML and see the changes live

(Obviously the changes don't stick)

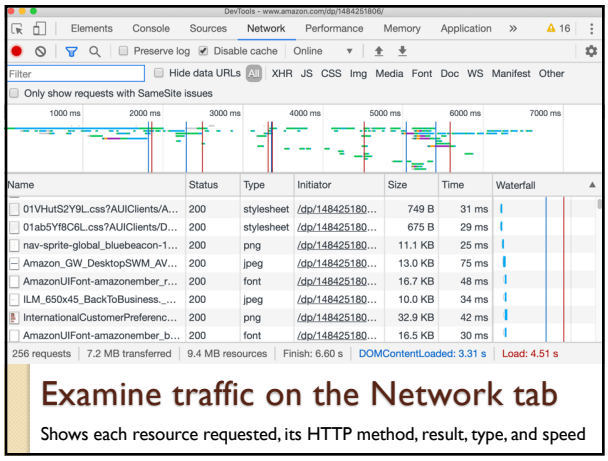
36

Debugging HTML and CSS

37


Working with CSS in the Computed tab

38


Examine traffic on the Network tab
Shows each resource requested, its HTTP method, result, type, and speed

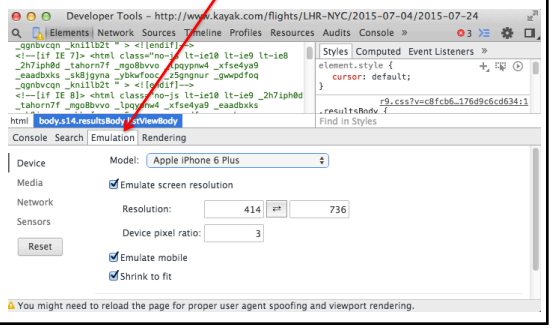
39


You can simulate mobile devices

40

Then pick your preferred device

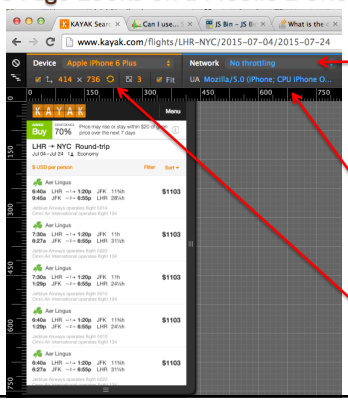
Choose your device under the Emulation tab



41

How the Page Looks on a Mobile Device

Note: you're still on the desktop browser. It just looks like a mobile browser.



Simulate a slow network

Choose your mobile browser agent

Rotate the device

42

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

43

Page Setup

The subtleties of correct HTML!

44

```
<h1>HTML is the language of the web</h1>
<p>It conforms to certain rules</p>
<ul>
<li>Tags may nest but not overlap</li>
<li>Tags must be closed or self-closing</li>
<li>They may have attributes which ...
  <ul>
    <li>may have quoted values</li>
    <li>are usually kebab-cased</li>
   <li class="attributes alter"></li>
  </ul>
</li>
</ul>
```

47

Laying out a document

48

Good HTML has four elements

- DOCTYPE
- html
- head
- body

49

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>
```

50

The <html> tag wraps the entire page

51

The <head> tag contains the metadata for the page

- <script>
- <style>
- <title>
- <meta>

52

The <body> tag

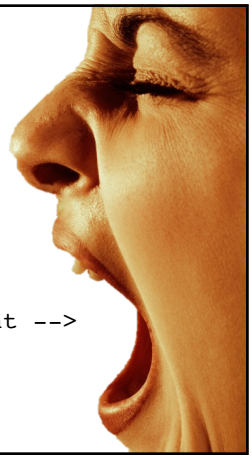
53

Altogether it may look like this

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

54

Commenting your HTML

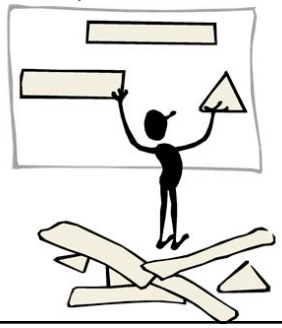


```
<!-- This is a comment -->
```

55

There are many other tags for layouts

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



56

Displaying text

57

The tag is for phrasing content (like text)

- …
- Creates a line break before and after

58

 creates a line break

- It is a singleton tag

```
<p>
Kreese: Sweep the leg.<br />
[Johnny stares at him in shock]<br />
Kreese: Do you have a problem with that?<br />
Johnny Lawrence: No, Sensei. <br />
Kreese: No mercy.
</p>
```

59

Headings can go from levels 1 to 6

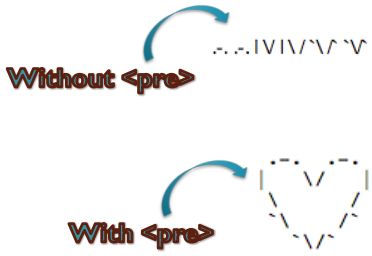
```
<head>
<title>Greendale Classes</title>
</head>
<body>
<header>Choose your classes</header>
<h1>Class List</h1>
<h2>College of Business</h2>
<h3>Finance Department</h3>
<h4>Undergraduate classes</h4>
<h5>Business Valuations 101</h5>
<h6>Section 114</h6>
```

60

Preformatted text

- HTML usually collapses whitespace
- <pre> preserves it

```
<pre>
 .-.  .-.
|   \/   |
 \       /
  `\   /`
    `\/`
</pre>
```



61

There are special characters

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

© < > & ® °

63

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>
```

64

Have different types of ordered lists

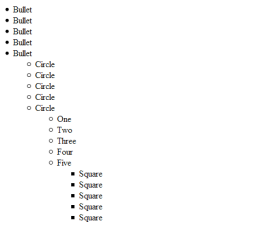
```
<ol style="list-style-type:_______">
```

- decimal (default)
- upper-alpha
- lower-alpha
- upper-roman
- lower-roman
- … and more

65

You can have different bullets with unordered lists

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



66

Just a note on Unicode

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

74

Displaying links

75

The anchor tag allows hyperlinking

- Allows linking from section to section and page to page.
- Link to another page:

```
<a href="http://www.othersite.com">Go to other site</a>
```

- Link to someplace on this page:

```
<a href="#aParagraph">Go to another paragraph.</a>
```

- … and …

```
<a id="aParagraph"></a>
```

76

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>
```

77

Displaying images

79

Add images with <img>

- Inserts an image.

```
<img src="logo.jpg" />
```

- Formats allowed: jpg, gif, png
- Can specify sizes and alternate texts:

```
<img src="logo.jpg" alt="Company logo"
height="52px" width="100px" />
```

- alt text for missing/jank images and vision-impaired surfers
- Can make it a link, too:

```
<a href="tic.com"><img src="logo.jpg" /></a>
```

80

New! Lazy Loading

```
<img src="foo.jpg" loading="lazy" />
```

- 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.

81

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.

82

The title attribute

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

- Generally avoid it.

83

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 <ol>, <ul> and

85

Effective CSS Styling

How to fine-tune your page's look

88

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

90

91

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!

92

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;
}
```

93

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

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

94

… Or on the page …

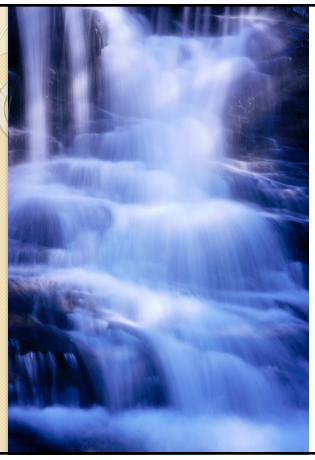
```
<head>
<title>I &lt;3 Styles!</title>
<style type="text/css">
.headlines, .sublines, .infotext {
  font-face: arial;
  color: black;
  background: yellow;
  font-weight: bold;
}
.headlines {font-size:14pt;}
.sublines {font-size:12pt;}
.infotext {font-size: 10pt;}
</style>
</head>
```

95

… or for the entire site

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

96



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

97

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")

98

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

100

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

101

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

102

This is !important

- You can prevent overriding with *!important*

```
div {
  background-color: white !important;
}
```

Use with caution! Makes it tough to debug.

103

... like variables in your CSS

CSS custom properties

104

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;
}
```

105

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);
}
```

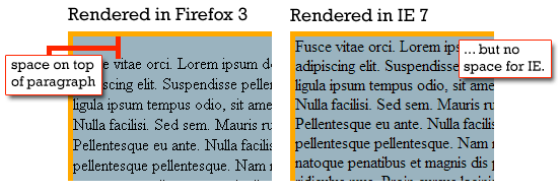
106

Resets

107

A plain, unstyled page looks different in different browsers

- Why?



108

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;
}
```

109

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

110

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

111

Semantic Grouping

So they can be positioned and styled as a unit

116

We need groupings to specify areas of our pages.

117

```
<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

118

Elements come in two flavors: block-level and inline

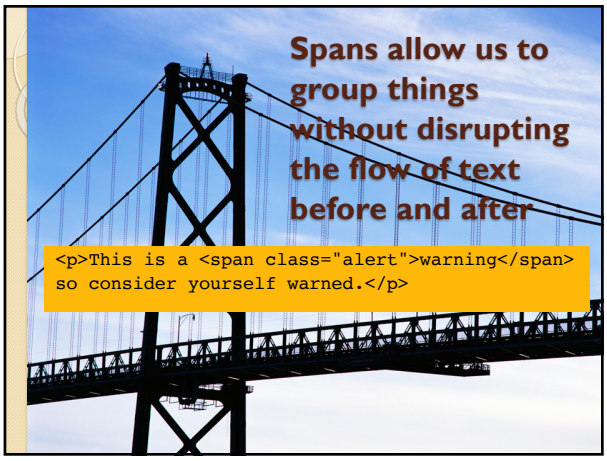
Block-level

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

Inline

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

119

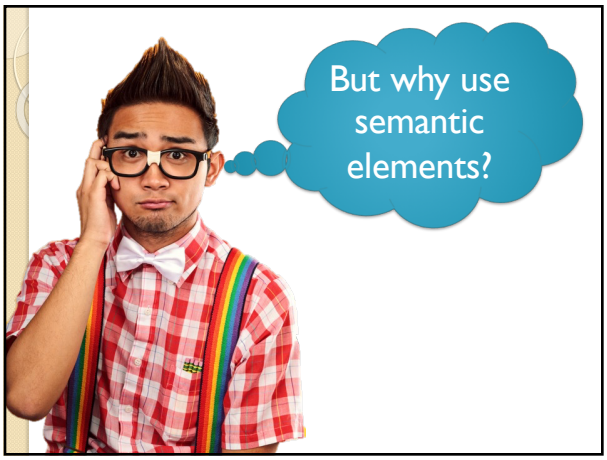


120

HTML5 semantic elements communicate meaning

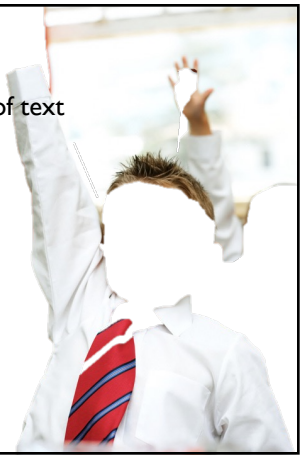
- section
- nav
- article
- aside
- header
- footer
- main
- details
- summary
- figure
- hr

121



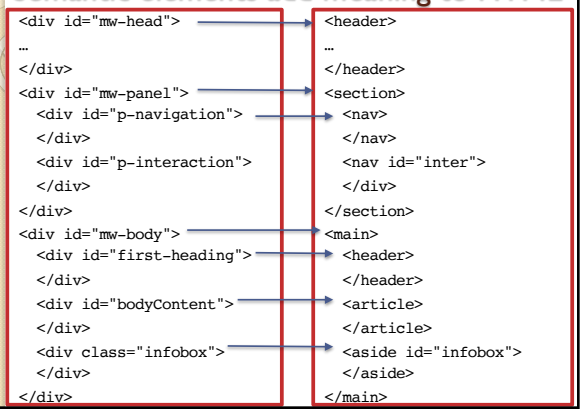
122

<p> tags hold ...
... a generic paragraph of text
 tags hold ...
... list elements
<img> tags hold ...
... pictures, photos
 tags hold ...
... uh ... ummm ...



123

Semantic elements add meaning to HTML



124

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?!

125

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

126

A section sounds like a div, right?

<section>

- Generally would appear in an outline of a page

** **

- 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

127

Articles are for content that could be syndicated

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

128

The differences between sections and articles are subtle

Use an article if …	Use a section if …
It is self-contained	It is **part** 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

129

A nav is a section with links to other pages or to parts within the page

- Site maps
- Those sections often found in the left column

130

131

Oh, by the way … an aside is a section that is related to the main content

- "The aside element represents a section of a page that consists of content that is tangentially related to the content around the aside element, and which could be considered separate from that content. "

132

asides would have the same kinds of things as a sidebar in a printed magazine or newspaper

- Pull quotes
- Sidebars
- Advertisement sections
- Groups of nav elements
- Call to action

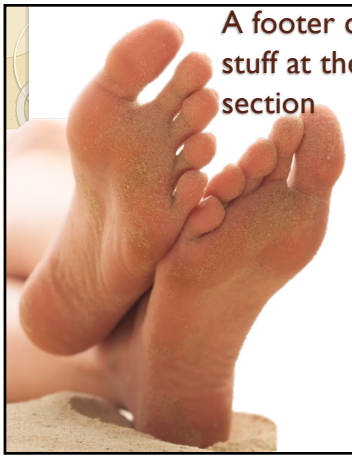
133

A header contains the stuff at the top of a section

- "The header element represents a group of introductory or navigational aids."

134

A footer contains the stuff at the bottom of a section



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

135

<main>

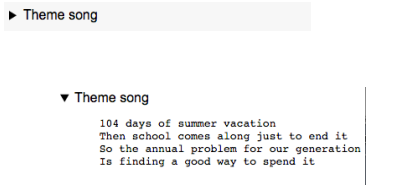
- Newest of the new tags
- 1 per doc
- Must not be a descendent of an <article>, <aside>, <footer>, <header>, or <nav> element.

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

136

<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>
```



137

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);
}
```

138

```
<figure>
  <img src="perry.jpg" />
  <figcaption>
    Perry (Agent P)
  </figcaption>
</figure>
```

Do this ...

... not this

```
<div class="figure">
  <img src="perry.jpg" />
  <p>
    Perry (Agent P)
  </p>
</div>
```

139

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

143

How to Position with CSS

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

146

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

148

Laying out sections of a page

The box model

149

The box model provides aesthetic space

Margin
- Space outside the container between sections

Padding
- Space inside the container between the container and the contents

150

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

151

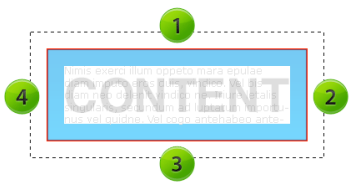
So, which do I use?

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

154

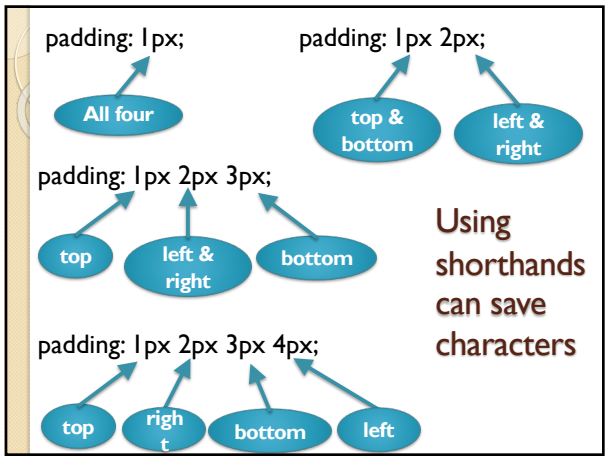
You can have different box sides

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

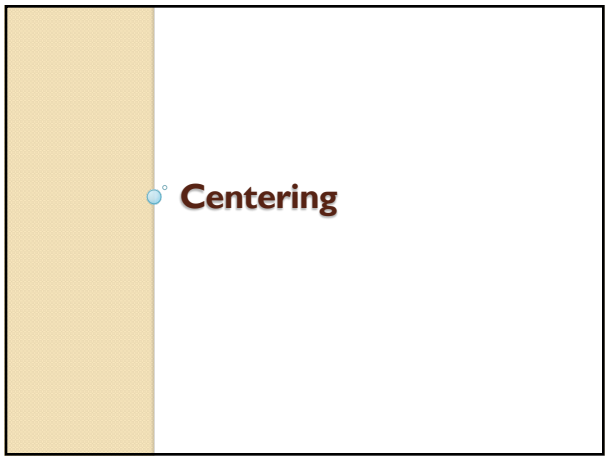


Top, Right, Bottom, Left

156



157



163

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.

165

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.

166

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.

167

The position style

168

The position style has discrete values:

```
static
relative
absolute
fixed
```

(and a few others)

169

static (the default)

- 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.

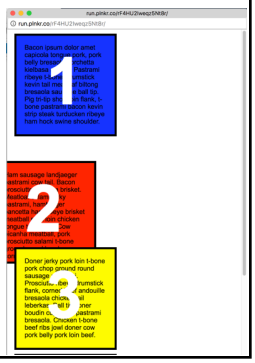
```
div:nth-child(2) {
  position: static;
}
```

170

relative

- 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;
}
```

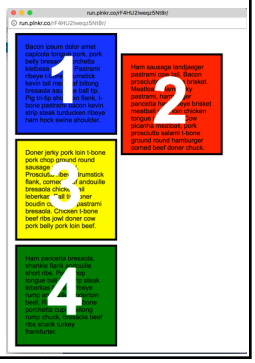


171

absolute

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

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



172

fixed

- 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;
}
```

173

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

175

Page Layout Strategy

What are my options and which do I choose?

178

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?

181

How do you layout pages?

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

182

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

- Tables
- Absolute positioning
- Inline sections

183

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

187

Four ways to hybrid *inline* and *block*

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

188

Option 1

Floated Divs

189

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;`	`clear: both;`
Says "I'm going to move left. Let the thing below me float up on my right"	Says "You below can't float higher than me"
float: right does the same but moves to the right	re-establish breaks

190

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	With float
div1 div2 div3 div4 div5	div1 div2div3div4div5

191

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

192

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

193

Option 2

display: inline-block

198

The good ...

- display: inline-block has the best of both worlds
- Honors width like a block
- Allows side-by-side like an inline

199

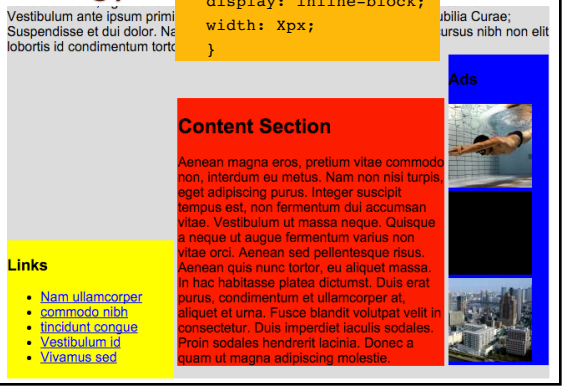
The bad ...

- Alignment and spacing are issues
- They're vertically aligned at the bottom :-(
- There's a space between sections even with no margin. :-(

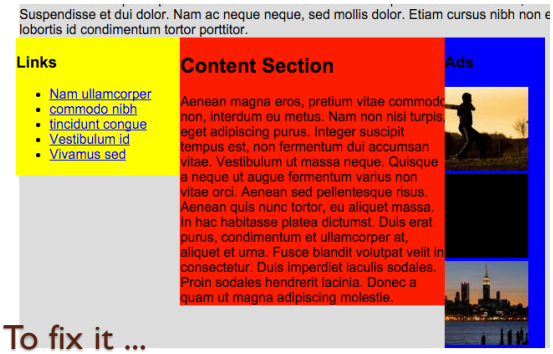
200

The ugly ...

```
section {
  display: inline-block;
  width: Xpx;
  }
```



201



To fix it …

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

202

A very high-level intro to flexbox and grid

203

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

205

Grid

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

206

So ... which one do I learn?

float vs inline-block vs grid vs flexbox

207

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

208

Indeterminate number of items

- All of the layout options handle extra items well.
- Flexbox can wrap and it will wrap as long as it needs to.
- Grid will repeat the last row as many times as needed.
- Float wraps also.

211

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?

212

Layouts with Flexbox

The deep dive into CSS flexbox

219

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

220

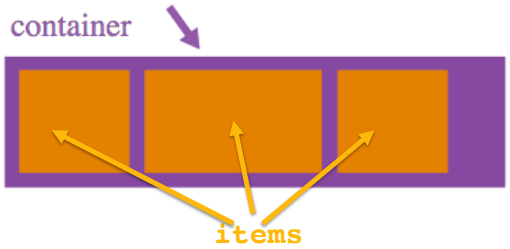
Option 3

Flex boxes

221

222

flex involves a container and contents



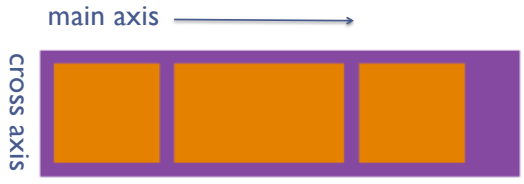
223

Mark the container as a flex-box

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

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

224



The container has axes

Most of us would call these

- horizontal
- vertical

225

The container lays out the items

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

226

Are we going to wrap or not?

Two ways to think about flexbox ...

227

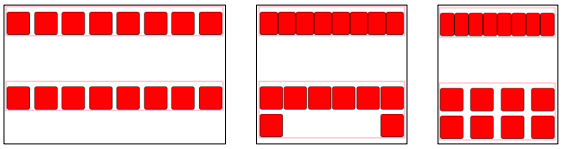
Two ways to think about flex

- 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

229

Think of flexbox like this ...

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



230

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.

231

Flexbox - no-wrap

232

flex items have sizes

They have a "favorite" size

flex-basis

They know how to shrink from that and how to grow from that

flex-shrink
flex-grow

234

Sizes of the items

- flex-basis = the item's "favorite" size.
  - 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

235

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.

237

order

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

238

Wrapping with flex

239

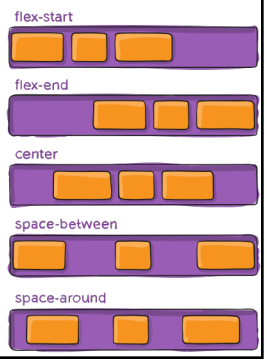
flex-wrap tells them to wrap when needed

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

240

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

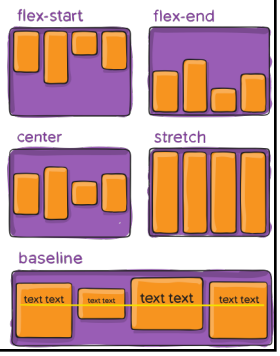
And the container controls the spacing



241

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

And the container controls the spacing



242

Bonus! How to center anything

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

243

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

246

Layouts with Grid

The deep dive into CSS grids

251

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.

252

Option 4

Grids

254

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*

255

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

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

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

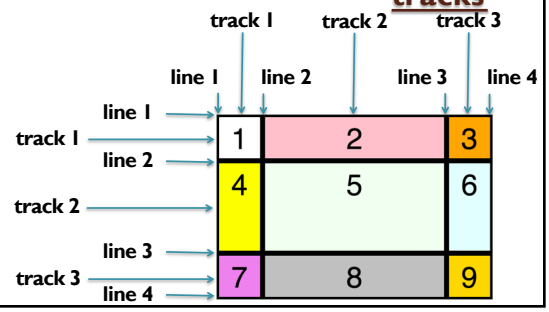
256

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

257

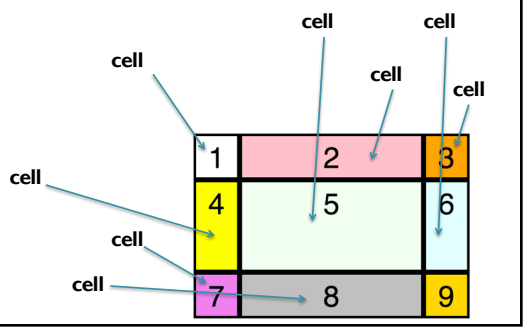
```
.grid {
  display: grid;
  grid-template-columns: 50px 200px 50px;
  grid-template-rows: 50px auto auto;
}
```

grid **lines** separate the grid **tracks**



258

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



259

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.

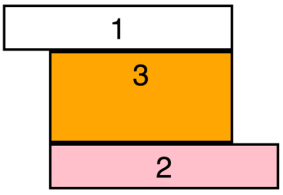
260

```
.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;
}
```

Fill the columns between line 1 and line 3

Fill the rows between line 1 and line 2

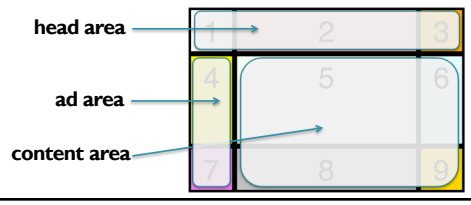
To assign to cells, place between **lines**



261

```
.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'
}
```

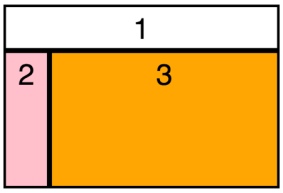
grid **areas** are optional groups of cells



262

```
.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;
}
```

To assign to areas ...



263

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.

265

Progressive Web Apps and Responsive Design

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

270

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

271

Programming for mobile devices is difficult

- iOS
- Objective-C
- Cocoa Touch
- XCode IDE

- Android
- Java
- ADT plugin
- Eclipse IDE

272

But if we create a web application, users can simply browse

273

Ummm ...This is really nothing new

So what are the implications?

275

Pros

1. One language, environment, IDE
2. Only one version of the code base exists
3. Super-easy deployment

276

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)

277

A 10,000 foot view

Progressive web apps

278

PWAs requirements that everyone agrees on

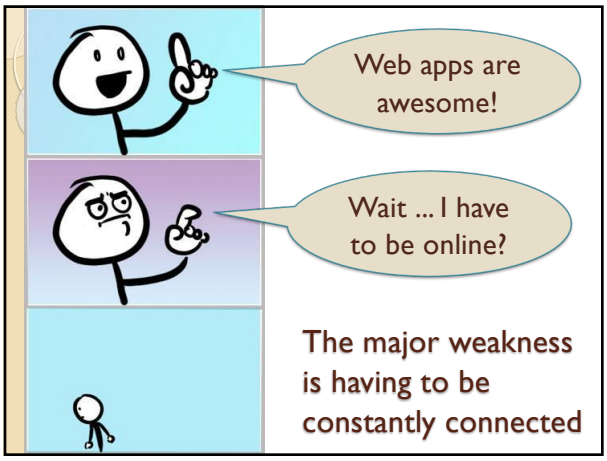
- Must be HTTPS
- Runs offline
- User can put an icon on the home screen

279

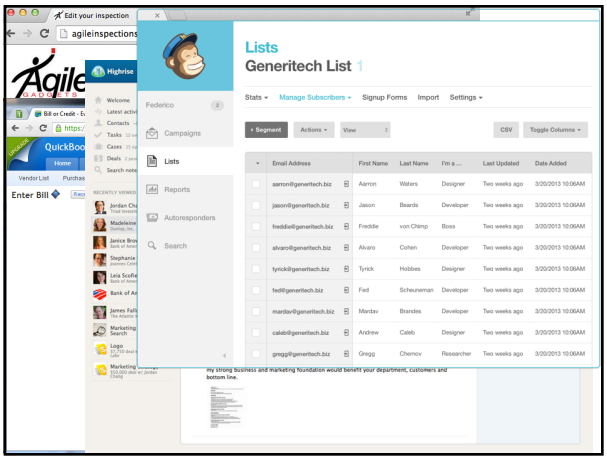
Some add these

- 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

280



281



282

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

283

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

284

To go offline, you have to have a manifest

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

285

```
{
  "short_name": "Maps",
  "name": "Google Maps",
  "icons": [
    {
      "src": "/images/icons-192.png",
      "type": "image/png",
      "sizes": "192x192"
    },
    {
      "src": "/images/icons-512.png",
      "type": "image/png",
      "sizes": "512x512"
    }
  ],
  "start_url": "/maps/?source=pwa",
  "background_color": "#3367D6",
  "display": "standalone",
  "scope": "/maps/",
  "theme_color": "#3367D6"
}
```

A manifest lists what is needed locally

286

Using responsive design to make each page look good on any screen or on paper

Responsive By Hand

300

- PCs
- Tablets
- Handhelds
- Books
- TV
- Printers
- Glasses
- Watches

301

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 ...

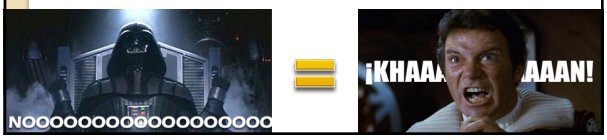
Responsive Design

302

Responsive & Adaptive both solve this problem

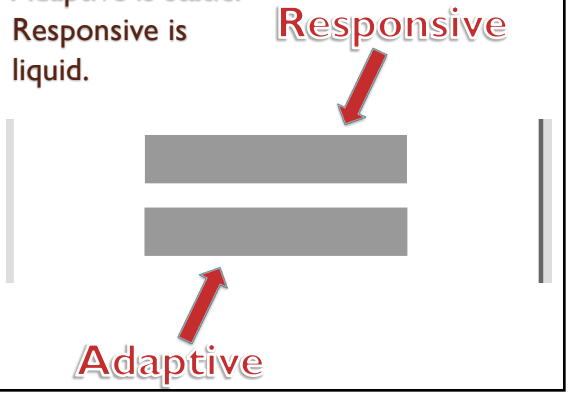
Responsive Design	Adaptive Design
• Sections resize with the width	• Sections stay the same width
• It is liquid within bands	• It is static within bands

They're essentially the same



303

Adaptive is static.
Responsive is liquid.



304

This is a basic tenet

305

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

306

The *media* attribute is one key

How it might look in our CSS

```
@media print {
  /* Print layout here */
}
@media screen {
  /* Screen layout here */
}
```

W3C-recognized types

- braille
- embossed
- handheld
- print
- projection
- screen
- speech
- tty
- tv

307

Width is the other key

308

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" />
```

309

What things can you look for?

- resolution
- orientation
- device-aspect-ratio
- color | monochrome
- width
- max-width
- min-width
- device-width
- max-device-width
- min-device-width
- all the above for height, too

310

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 */}
```

311

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

318

Modern CSS Formatting

How to make the web look great using modern techniques

322

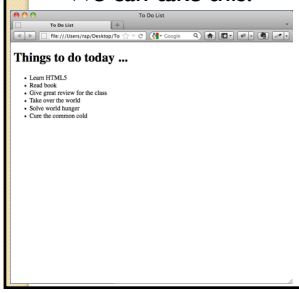
tl;dr

- 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

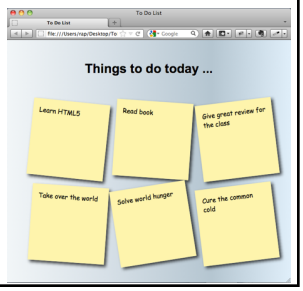
324

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

We can take this:



And make it look like this:



325

StylING Text

326

What properties can be set?

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

327

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

328

font-style

- font-style: normal|italic|oblique

```
font-style: italic;
```

330

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

331

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;
```

332

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;
}
```

333

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)

334

Here's an example

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

335

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.

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.

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 piece of information that requires special attention.

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.

336

text-decoration

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

```
text-decoration: underline;
```

337

Who doesn't like tips?

css Tips and Tricks!

339

Backgrounds

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

340

But what if the background image doesn't fit exactly in its container?

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

341

Top to bottom

```
background: linear-gradient(to bottom,
  white, red);
```

Left to right

```
background: linear-gradient(to right,
  red, white);
```

Bottom right to top left

```
background: linear-gradient(45deg,
  red, white);
```

Color gradients look cool!

342

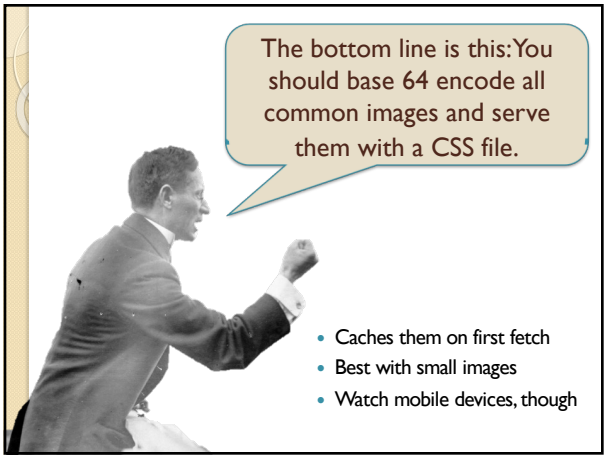
Data uris may speed up your images

```
.twitterLogo {
  width: 20px;
  height: 20px;
  background-image: url("...kSuQmCC");
}
```

Pros	Cons
• Fewer HTTP requests!	• Size is increased by 1/3 • Browser has to process the image

So what should we do?

345



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

346

CSS Filter property

Applies filters to images!

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

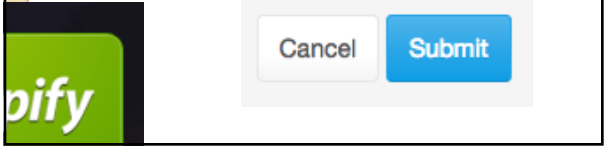
347

349

Rounded corners

border-radius: pixels;

```
.dialog {
  border-radius: 5px;
}
```



350

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

352

Advanced CSS Selectors

How to point to almost anything

354

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

356

Remember the basic style syntax

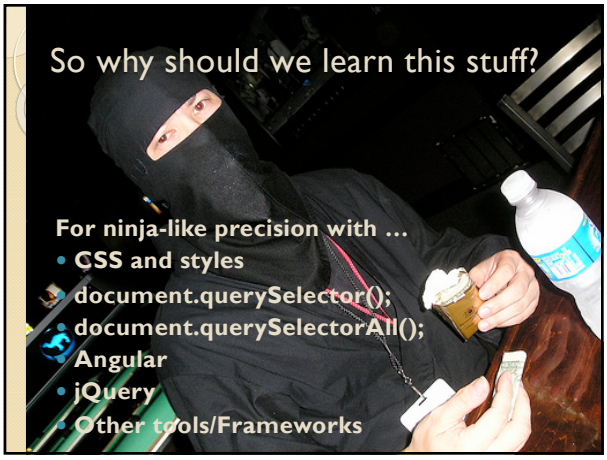
```
selector {
  property: value;
  [property: value …]
}
```

357

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
- …

358



359

361

Remember your basic selectors?

362

Element selectors point to all elements of a certain type

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

363

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

364

ID selectors point to exactly one thing

365

Combining selectors

366

Concatenating = all requirments

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

367

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

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

368

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;}
```

372

Selecting by DOM position

Relationship selectors

374

The DOM also points to relationships between elements

- Descendants
- Parent
- Child
- Siblings
- Ancestors

375

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;
}
```

376

Child selectors

`parent > child`

- Selects only **direct** children

```
div#sidePanel > p {
  text-decoration: bolder;
}
```

377

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;
}
```

378

Adjacent sibling selectors

`sel1 + sel2`

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

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

379

Attribute selectors

381

Remember XML attributes?

- Attributes are usually descriptors of an element.

```
<p class="fancy">
<a href="help.html" title="Get help">
<button value="Click me">
```

- You can select elements by HTML attribute

382

Attribute exists

- sel[attr]
- The attribute is *present*

```
img[alt]
a[href]
```

383

Attribute is *exactly* ...

- sel[attr="value"]
- The attribute is set to "value"

```
img[src="help.gif"]
```

384

Attribute starts with ...

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

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

385

Attribute ends with ...

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

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

386

Attribute contains ...

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

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

387

Pseudo-classes

388

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

389

Pseudo-classes for links

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

390

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

391

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

392

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
  - No siblings of the same type

- :nth-child(odd) or :nth-child(even) or :nth-child(3) or nth-child(5n + 1) or ...
- :nth-last-child(XXX) – counting from back
- :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.

393

nth-child(something)

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

394

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

395

Pseudo-Elements

396

Pseudo-elements

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

398

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

- Example

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

399

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

401

CSS3 Transforms and Transitions

How to create amazing effects and motion without JavaScript!

404

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

406

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>
```

407

CSS transforms can now alter DOM elements with styles

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

408

Scaling can be done through brute force

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

409

Scaling can be done by using transform

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

410

Location can be done by brute force …

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

412

… or by using transform

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

413

Rotation

```
img.verticalLogo {
    transform: rotate(-90deg);
}
```

- Angle in ...
  - deg
  - rad
  - grad
  - turn

415

Adding motion to your pages for fun and profit

CSS Transitions

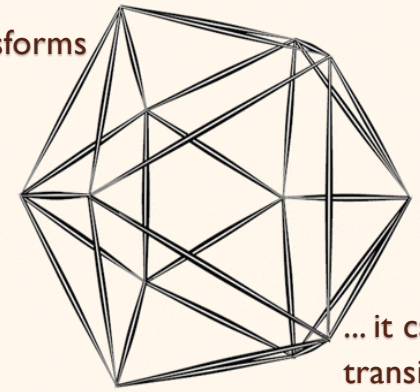
421

Intro to transitions

- The transforms section showed us how to change HTML elements
- Now let's see how to change them gradually

422

If it transforms ...



... it can transition

423

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

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

424

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>;
}
```

427

The property is the thing you want transitioned

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

… or …
- **all**

428

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

- In seconds or milliseconds
- 5s
- 5000ms

429

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)

430

The transition delay tells how long to wait before beginning

- in Seconds (s) or milliseconds (ms)

432

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;
}
```

433

We can combine transitions

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

434

... but this is not okay

```
#theDiv:hover {
  transition: top 3s;
  transition: left 1s;
}
```

- Why?
- Because the second overwrites the first.

435

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

439

Deep Dive into Tables

Presenting data in rows and columns

441

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

443

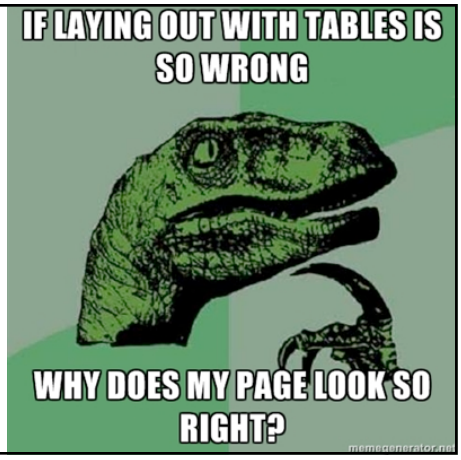
Sometimes data is best presented in tables

- 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

444

Tables are for data **ONLY**



445

Tables are simple to create, but verbose

<table> … </table>
<tbody> … </tbody>
<tr> … </tr>
<td> … </td>

446

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><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><td>
$23,812.27</td>
</tr>
<tr>
<td>Stanley
Hudson</td><td>$5,587.17</td><td>$9,493.05</td><td>$5,911.82</td><td>$
20,992.04</td>
</tr>
…
</tbody>
</table>
```

447

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></td><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>
```

448

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

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

450

Stylin' tables can and should be done using CSS

451

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;
}
```

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	$140,015.43

452

Alternating rows

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

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	$140,015.43

454

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	$140,015.43

455

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

457

Best Practices with Forms

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

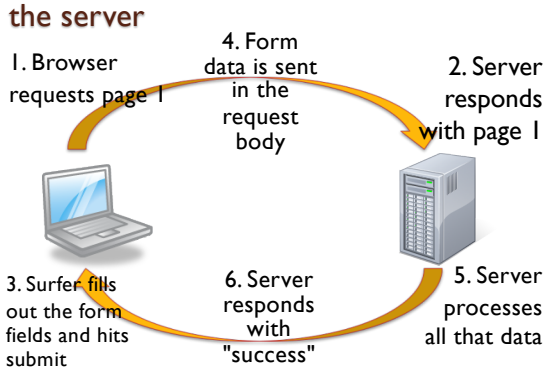
459

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

461

Forms are the main way to interact with the server



462

HTTP Methods

- GET
- HEAD
- POST
- PUT
- PATCH
- DELETE
- TRACE
- CONNECT
- OPTIONS

463

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"
}
```

464

The <form> tag

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

465

466

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

468

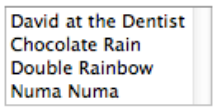
textarea is for multiline text in forms

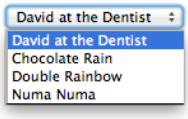
```
<textarea>
Digg is a social news website. Prior to Digg v4,
its cornerstone function consisted of letting
people vote stories up or down, called digging
and burying, respectively. Digg's popularity
prompted the creation of copycat social
networking sites with story submission and
voting systems
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

470

select can be listboxes or dropdowns






```
<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>
```

471

inputs are for everything else

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

- text
- password
- checkbox
- radio
- file
- submit
- button
- hidden

- email
- url
- number
- range
- date
- datetime
- month
- week
- time
- search
- color

476

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

478

Radio buttons

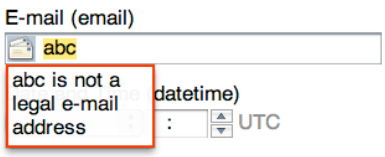
- To group them, give them the same name attribute.

```
<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>
```

479

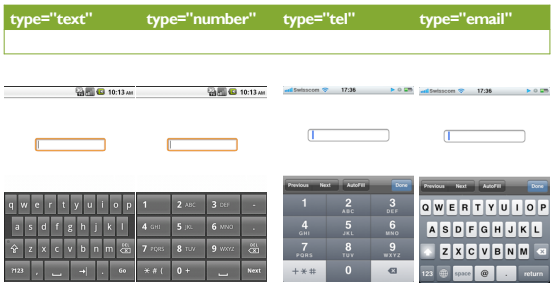
input type="email"

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



483

The keyboard changes with a type set



484

input type="number"

- A number

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

487

input type="range"

- Also a number

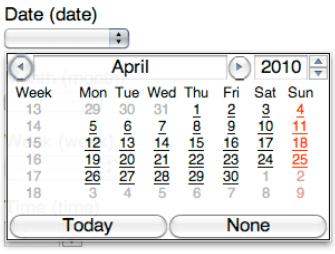
```
<input type="range"
       min="0"
       max="10"
       step="2"
       value="6" />
```

Range (range)

488

<input type="date" />

Date (date)



490

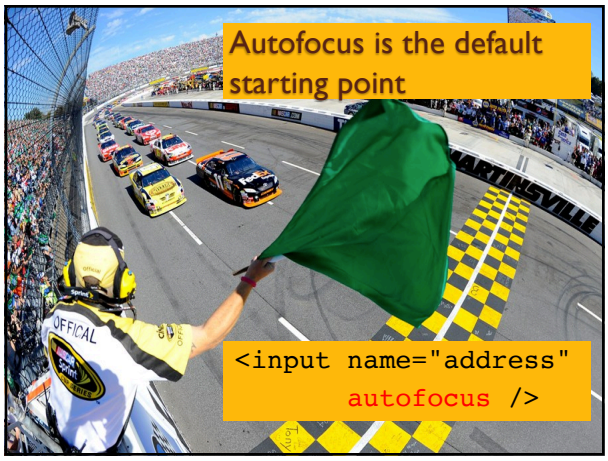
Form attributes

495

HTML5 attributes

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

496



497

Placeholder puts ghost text in the textbox

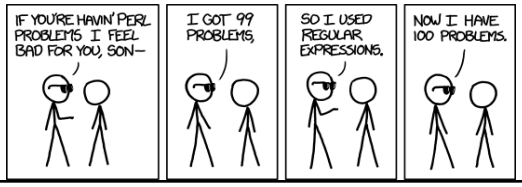
```
<input type='text' name='firstName'
placeholder='First Name' />
```

498

Built-in validation happens when you specify a pattern

- a regular expression

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



499

Requiring values

- Add "required" to any form element

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

500

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

506