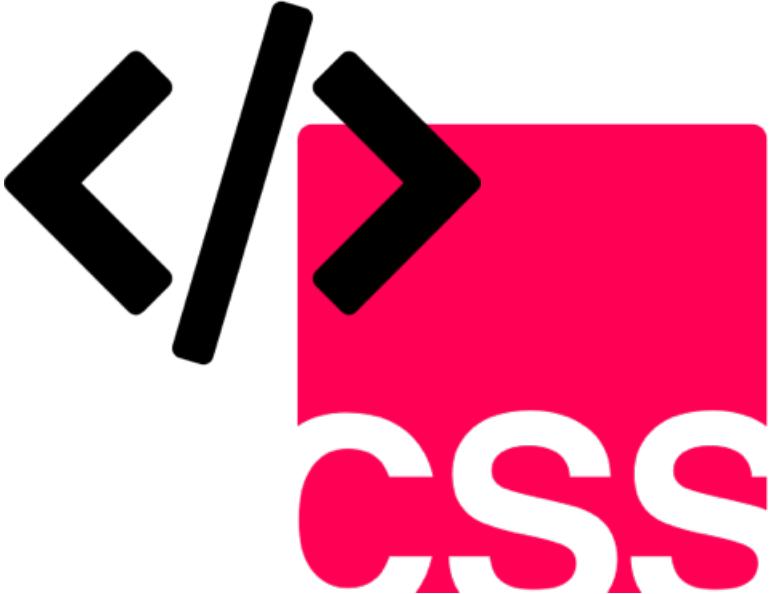


# How All This Actually Works



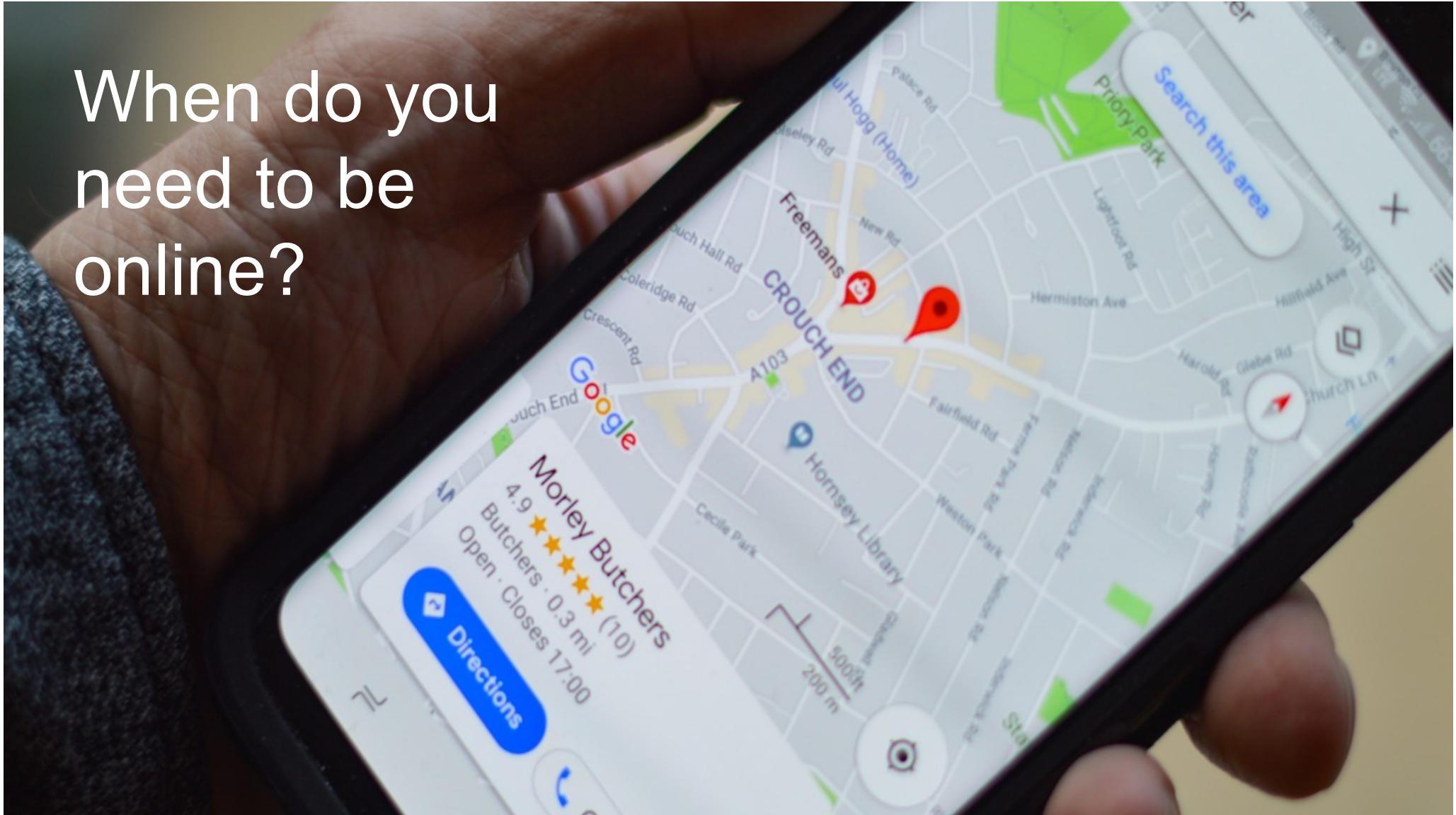
The word "css" is written in white lowercase letters on a red rectangular background.

The secrets behind the web

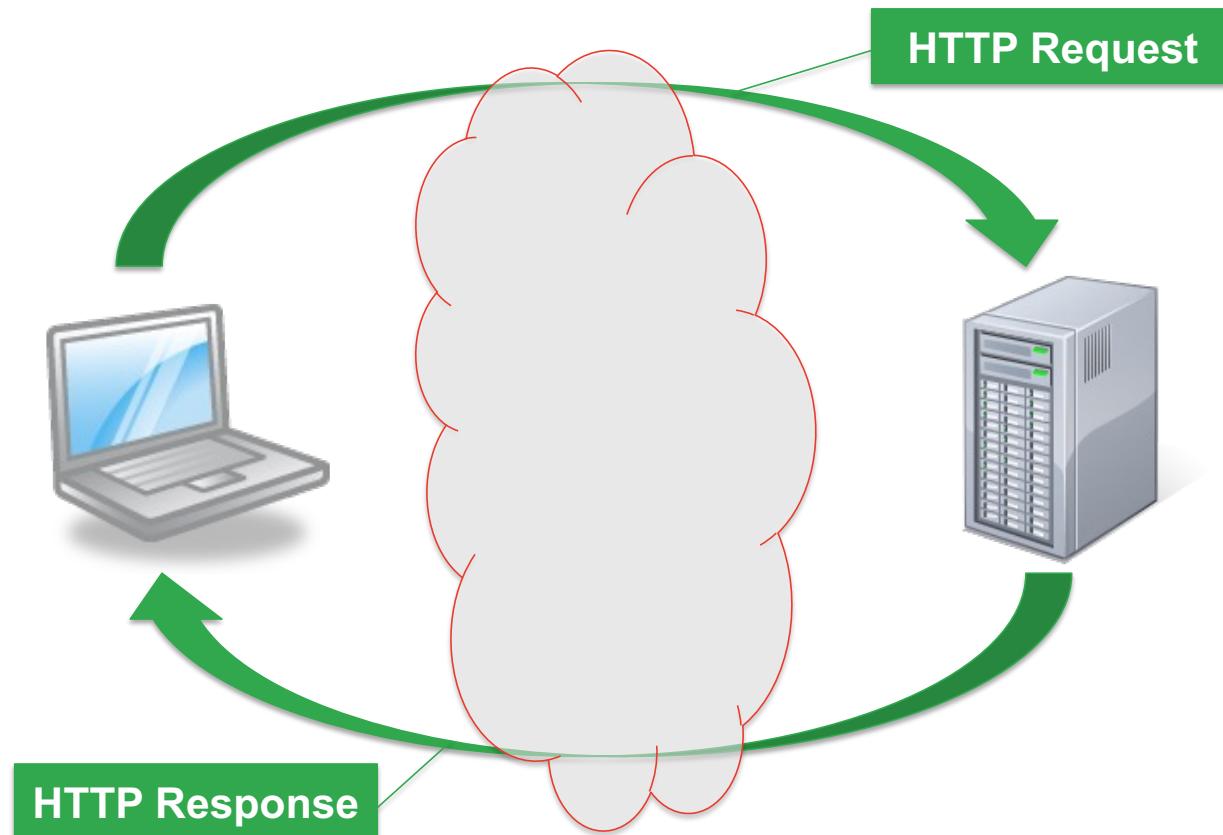
## tl;dr

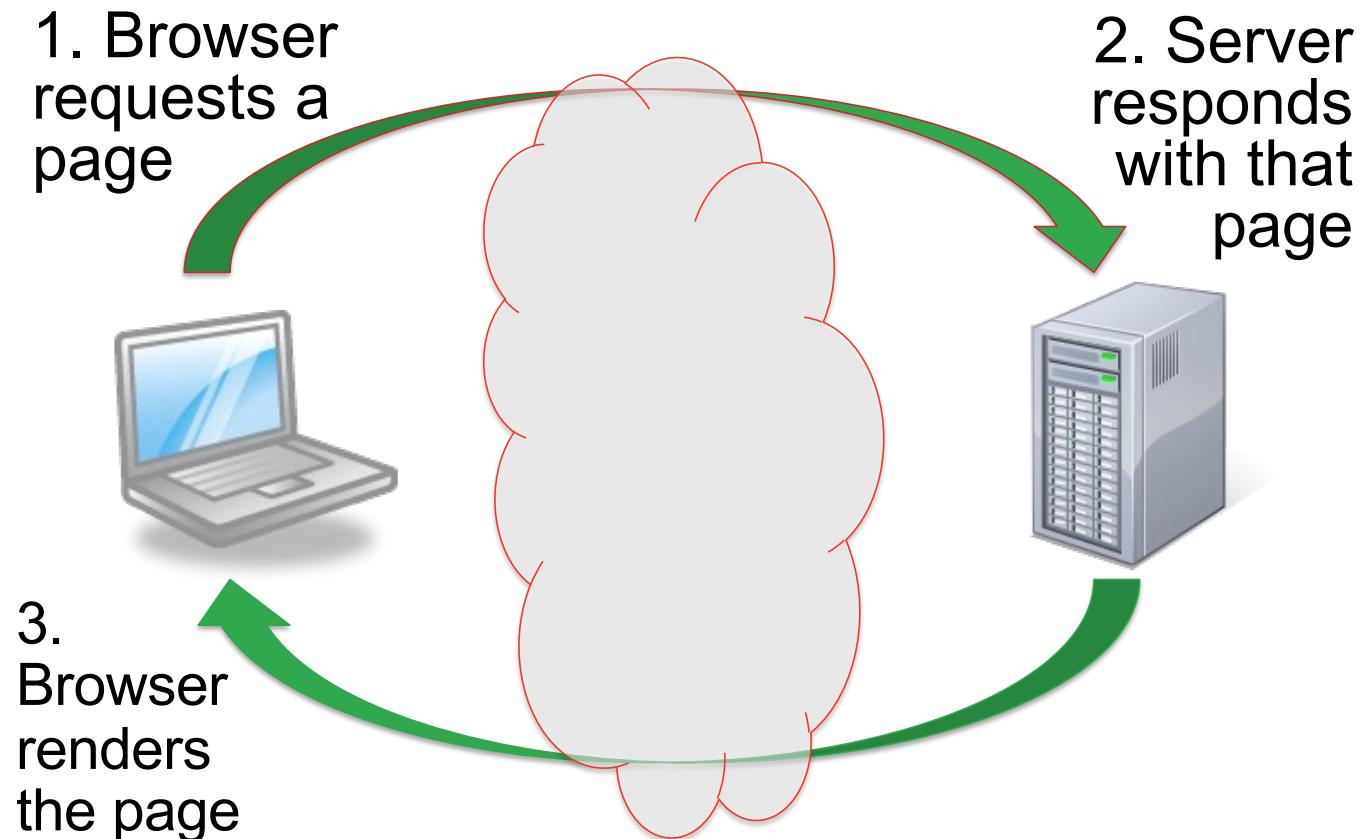
- Browsers run thin-client ... sort of
- They use HTTP and HTML, a static DSL
- They don't conform to a standard but the W3C/WHATWG makes one anyway
- The specs are not a great source of help. MDN and caniuse.com are excellent

# When do you need to be online?



# The web uses a thin client architecture





# Introducing \*ML

... because HTML is a markup language

# 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

# SGML is the original standard

```
Document: Bungler OED          At: "<entry>"  
  
<entry>  
  <hwsec>  
    <hwsp>  
      <hwlem>bungler</hwlem>  
      <pron>b<I>ʌŋ</I>ŋglər</pron>. </hwsp>  
      <vfl>Also <vd>b</vd> <vf>bongler</vf>.  
      </vfl>  
      <etym>f. as prec. + <xra><xlem>-ER</xlem>  
      </etym>  
      <sen>One who bungles; a clumsy unskillful person.  
      <quot>  
        <qdat>1533 </qdat>  
        <auth>MORE </auth>  
        <wk>Answe. Payson. Bk. </wk>Wks. (1557)  
        <qtxt>He is euen but a very bungler.  
      </quot>  
    </hwsec>  
</entry>
```

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

# 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



**XML**

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

**It needs to be**

- Well-formed**
- Valid**

## 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

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

- Okay

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

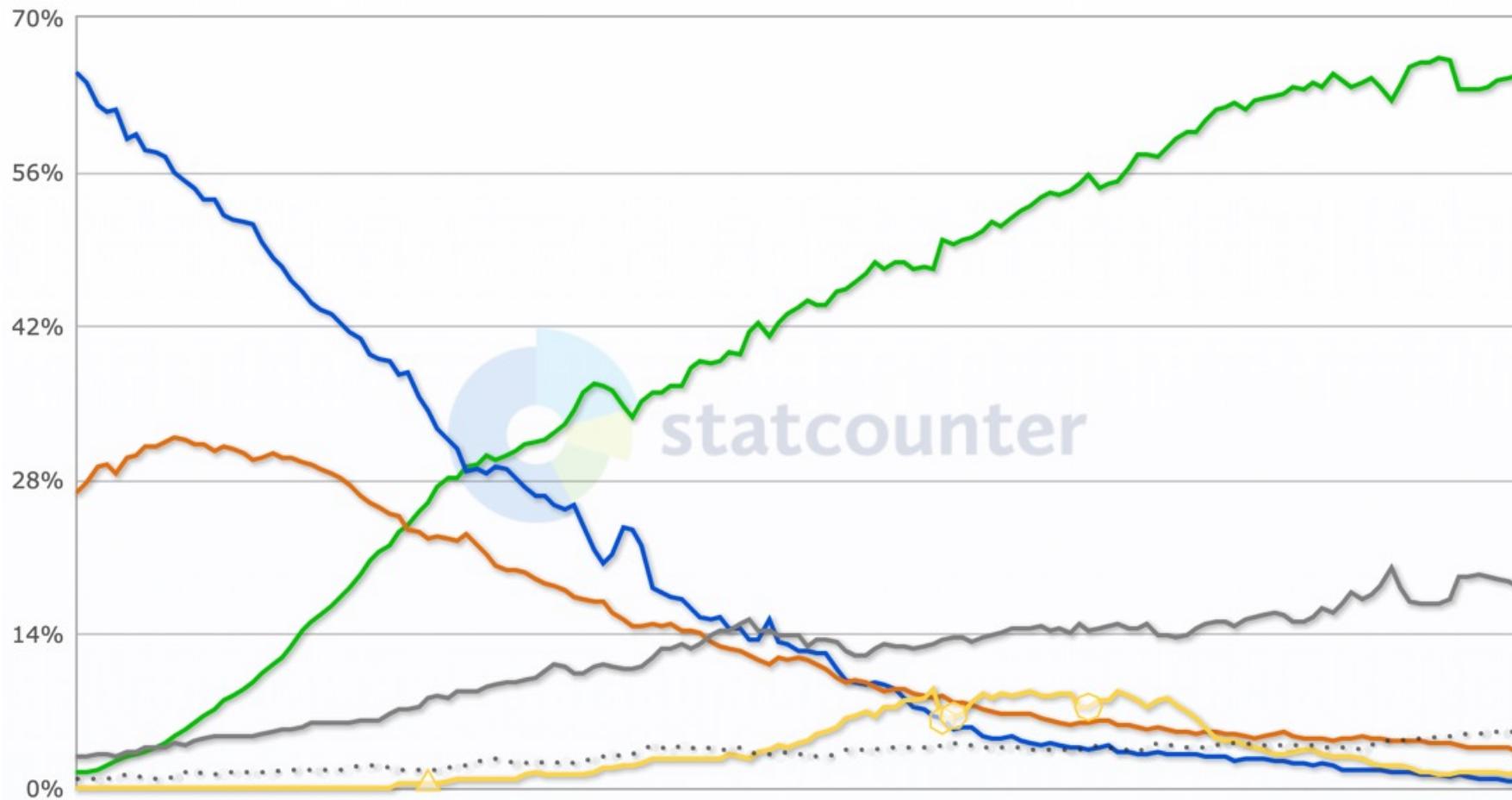
- Not okay

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





Different browsers lead to different renderings



And the popularity of browsers varies



So which browser do we build for?

**Best viewed with**



**Best viewed in Firefox**

**Best Viewed In:**



# 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

## How a specification is created

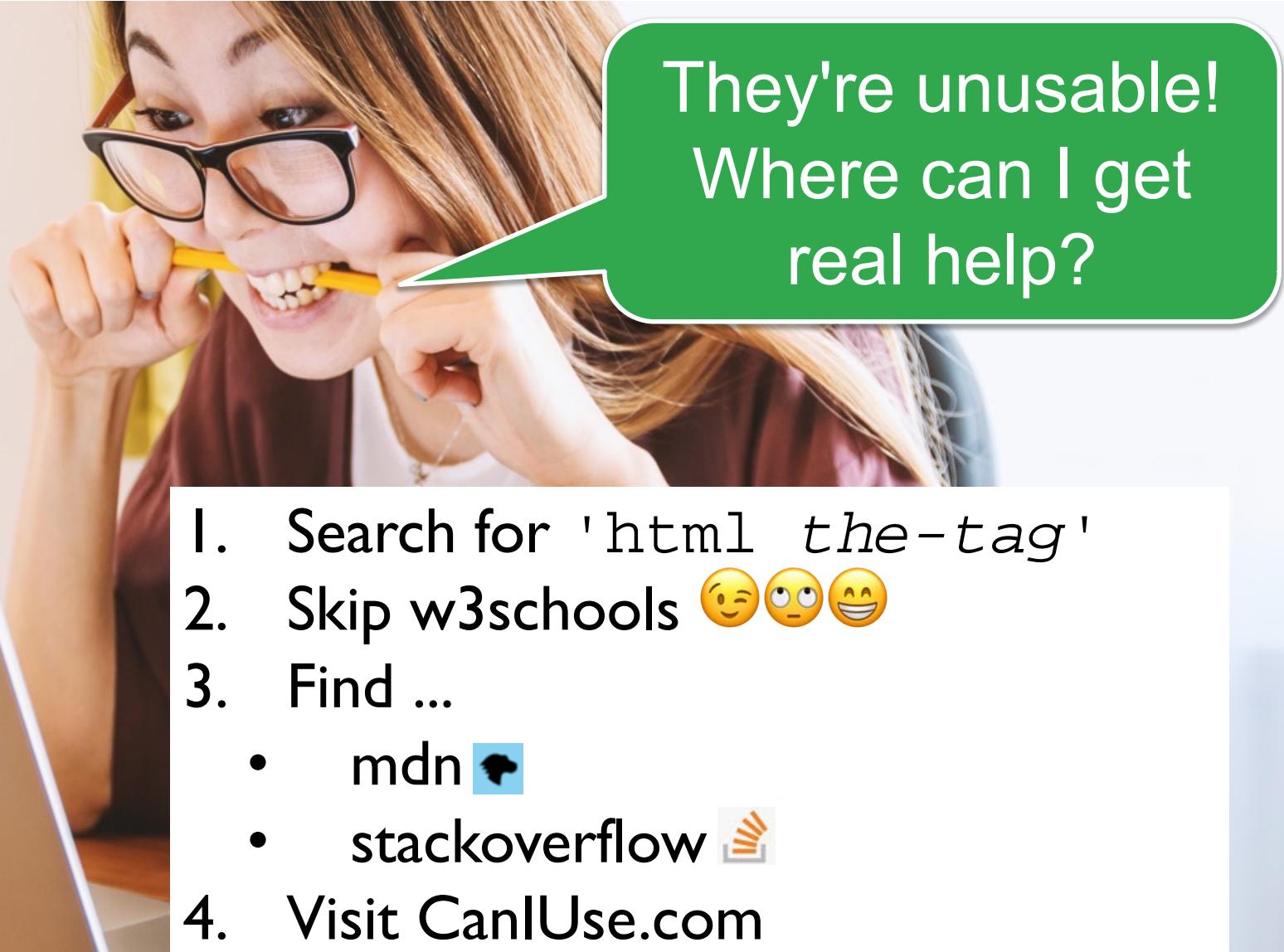
1. Developers unofficially come to a consensus
2. Someone ships code that works
3. Other browsers implement that same feature
4. The W3C & WHATWG make it official
5. All browsers eventually support it





<https://html.spec.whatwg.org/>

<https://www.w3.org/TR/>



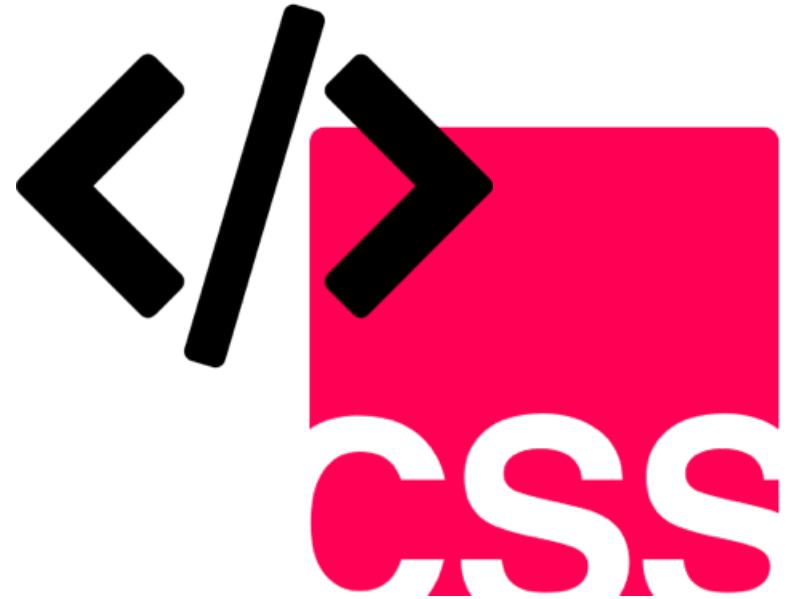
1. Search for 'html the-tag'
2. Skip w3schools 😊👀😊
3. Find ...
  - mdn 
  - stackoverflow 
4. Visit CanIUse.com

## 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

# Page Setup

The subtleties of correct HTML!



## 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 <i>

## foo.html

```
<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>
```

# Laying out a document

Good HTML has  
four elements



<DOCTYPE>  
<html>  
<head>  
<body>

# 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>
```



The <html>  
tag wraps  
the entire  
page

The `<head>` tag contains the metadata for the page

- `<script>`
- `<style>`
- `<link>`
- `<title>`
- `<meta>`



At minimum, the <head> should contain:

```
<head>
  <meta charset="utf-8" >
  <meta name="viewport"
    content="width=device-width, initial-scale=1" >
  <title>This is a web page</title>
</head>
```



The <body> tag

</head>

<body>

**Contains all of the markup  
that is displayed on the page.**

The <head>  
holds things that  
DON'T get  
rendered

```
<title>  <style>
          <script>
          <link>
<meta>
```

The <body>  
holds things that  
DO get rendered

```
<ol>      <td>  <a>
<form> <input> <li>
        <div>      <section>
<h2>          <em>
<p>           <h3>  <table>
<img>          <h1>  <code><ul>
<tr>
```

Tags can have **attributes** which further describe the tag



**Almost always  
key="value"**

```
<a href="foo.html">Foo</a>

<input type="text" name="ssn" required />
<label for="ssn">Social</label>
```

# The most common attribute? Id and classes to point to that element for JavaScript and CSS

```
<foo id="someId" class="class1 class2" />
```



Ids are unique  
on a page ...



... but there can be many  
elements with the same class

# Altogether it may look like this

## foo.html

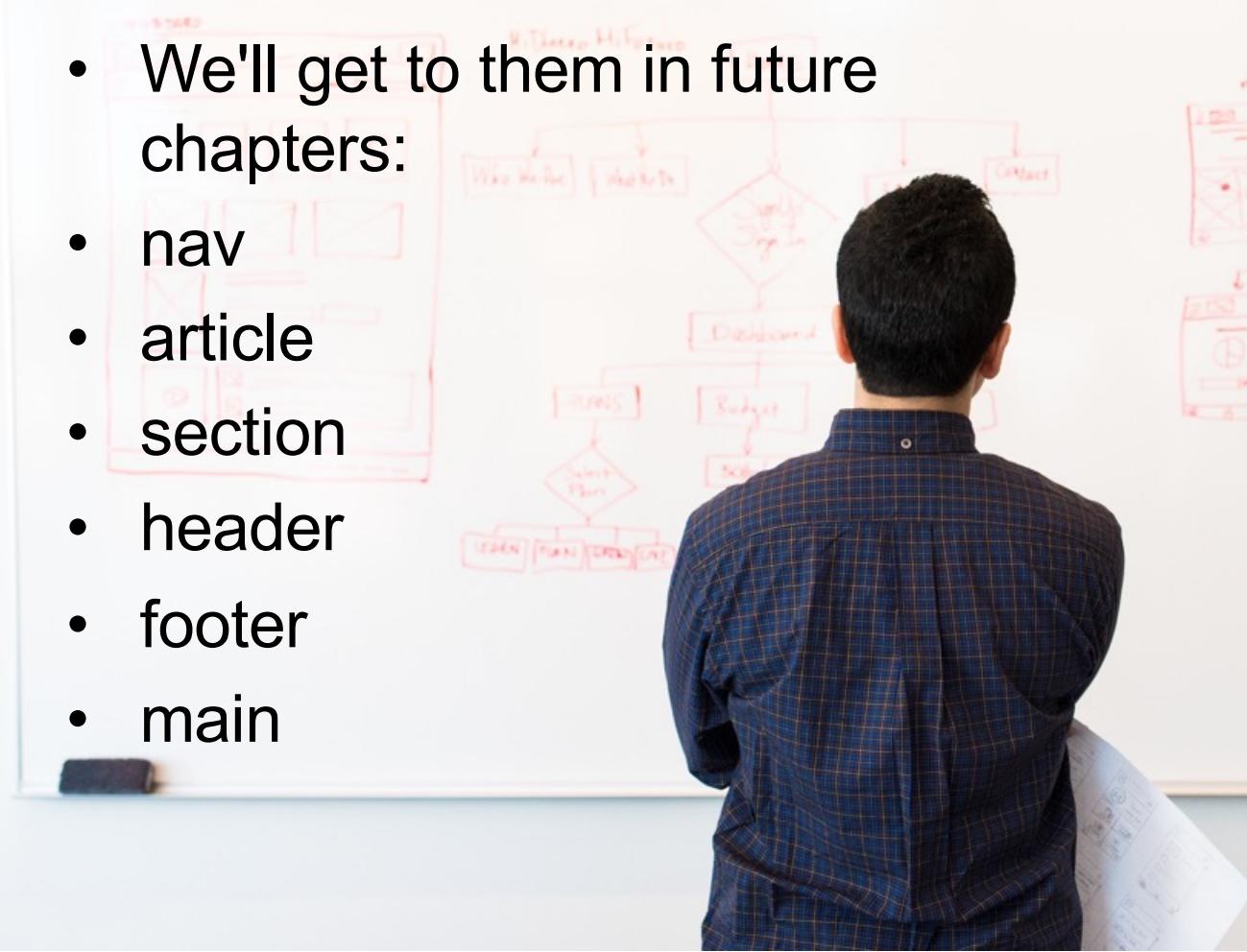
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport"
      content="width=device-width, initial-scale=1">
    <title>This is a web page</title>
  </head>
  <body>
    <!-- Most elements go here -->
  </body>
</html>
```

# Grouping things

## There are many tags for layouts

- We'll get to them in future chapters:
- nav
- article
- section
- header
- footer
- main

The two seminal ones are  
`<div>` and  
`<span>`





### **foo.html**

```
<div id="area1">  
    <!-- Other tags go here -->  
</div>  
  
<div id="area2">  
    <!-- Other tags go here -->  
</div>
```

A `<div>` is a division or area of a page

A silhouette of a suspension bridge against a clear blue sky with a few wispy clouds. The bridge's towers and cables are clearly visible.

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

<p>This is <span class="alert">an important warning</span>! Don't hit the button more than once.</p>

# Displaying text

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

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

## <br /> 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>

# 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>
```

# Preformatted text

- HTML usually collapses whitespace
- <pre> preserves it

```
<pre>
```

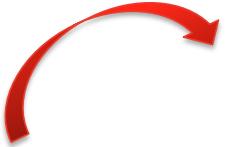
```
.-.-. .-.-.  
| \ / |  
 \ / /  
 `` /`  
 ``\``  
 </pre>
```

**Without <pre>**



```
...IVIV\`V`V
```

**With <pre>**



```
[-.-. .-.-.  
| \ / |  
 \ / /  
 `` /`  
 ``\``]
```



## Just a note on Unicode

UTF-8 can handle most real-world character sets like ...

Arabic

Cyrillic

Hebrew

Kanji

Etc.

## There are special characters

&nbsp;

&copy;

&lt;

&gt;

&amp;

&reg;

&deg;

© <> ® °

# 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>
```

# Displaying links

# 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>
```

# 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>
```

# Displaying images

# Add images with <img>

```

```

An actual file

Required for a11y

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>
```

## New! Lazy Loading

```

```

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

## 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
  - o guy in leather jacket with boots on table
  - o woman late for class
  - o man in apron preparing a baloney sandwich
5. Don't say it's an image or photo.

# The title attribute

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

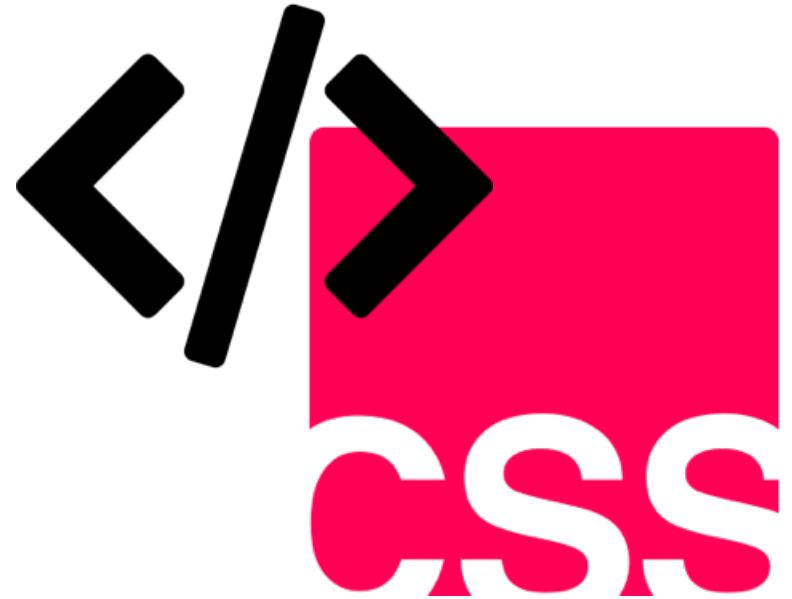
- Generally avoid it.



## 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 <i>

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

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



# All browsers have developer tools

- Most users never see them (or never care)
- Learn one, you've learned them all.
- Slight differences between terminology and placement

The screenshot shows the Chrome Developer Tools interface. On the left is the 'Elements' panel, which displays the DOM tree and the CSS styles applied to each element. A yellow rectangular overlay covers the central content area where the slide content would be displayed. The right side of the interface includes the 'Sources', 'Console', and 'Network' tabs, along with various developer settings and a search bar.

```
element { }
body { 
  font: normal 1em/1.2 sans-serif;
  background-color: #f0f0f0;
  z-index: 1;
  min-height: 100%; 
}
body, div, dl, dt, dd, ul, ol, form, fieldset, input, button, select, textarea {
  margin: 0;
  padding: 0;
}
<div id="onboarding-splash" style="display: none;>
  <div id="header" role="banner">...</div>
  <div class="side">...</div>
</div>
```

element.style {

```
> <div id="gtm-jail" name="{"subreddit": "", "origin": "https://old.reddit.com", "url": "https://old.reddit.com/", "userMatching": false, "userId": false, "advertiserCategory": ""}>...</div>
z-index: 1;
min-height: 100%;
```

}

body, div, dl, dt, dd, ul, ol, form, fieldset, input, button, select, textarea

reddit.dY1y2rRNqy0.css:1

But all browsers have debuggers baked in



**Option -  
Cmd - i**

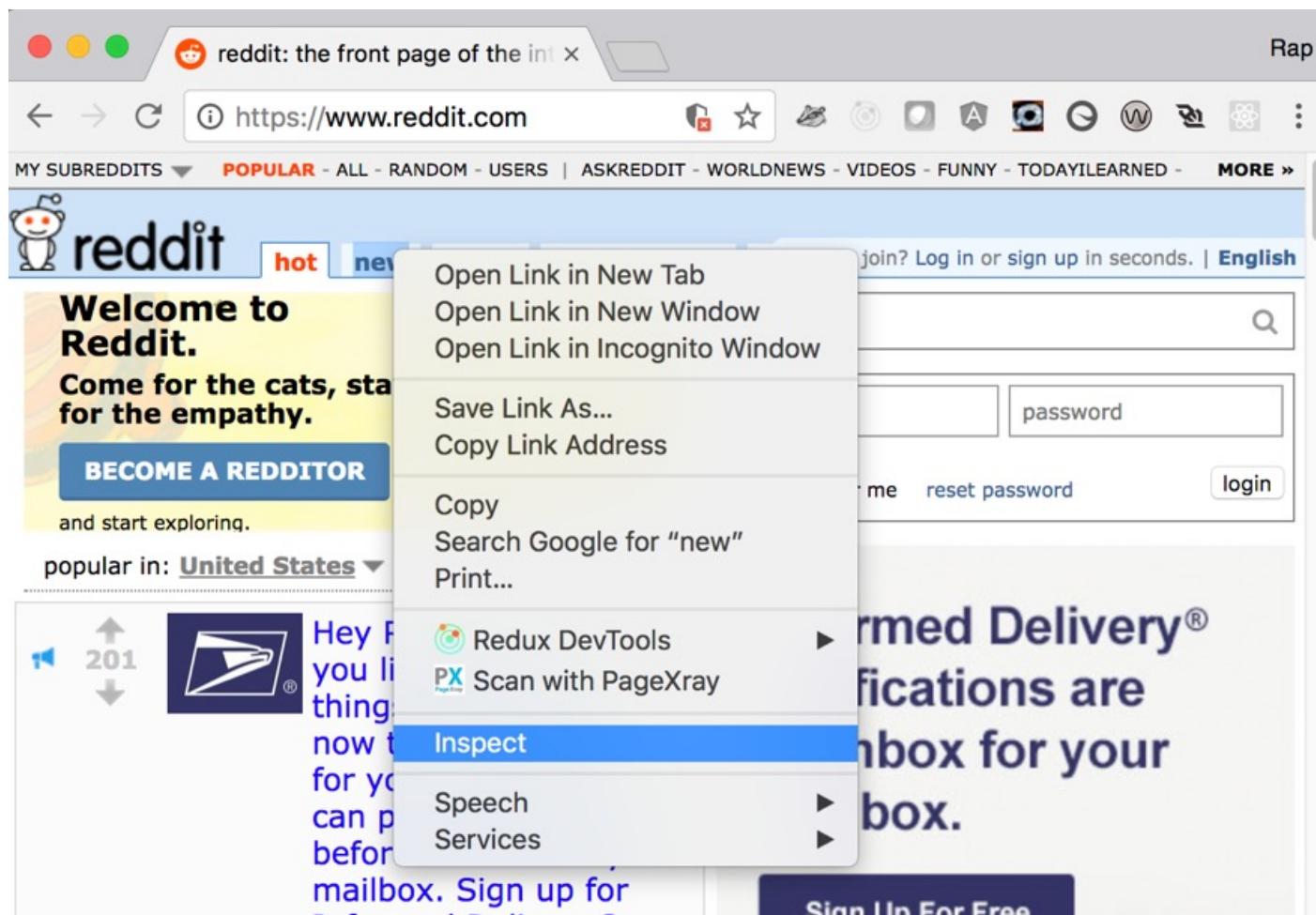


**F12**



No need  
for installs  
and extra  
packages

# ... Or you can choose 'inspect'



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

(Obviously the changes don't stick)

The screenshot shows the Google Chrome DevTools interface. The title bar reads "DevTools - www.imdb.com/title/tt0096697/?ref\_=fn\_al\_tt\_1". The left pane displays the DOM tree for the page. The right pane shows the "Styles" tab, which lists CSS rules applied to the selected element. The selected rule is ".article" from "title-flat-...356\_.css:28".

```
><script>...</script>
<div class="article" id="titleCast">
  <span class="rightcornerlink">...</span>
  <h2>Cast</h2>
  <table class="cast_list">
    <tbody>
      <tr>...
        <td class="primary_photo">...</td>
        <td>
          <a href="/name/nm0144657/?ref_=tt_cl_t1"> Dan Castellaneta </a>
        </td>
        <td class="ellipsis">
          ...
        </td>
        <td class="character">...</td>
      </tr>
      <tr class="odd">...
      <tr class="even">...
      <tr class="even">...
      <tr class="odd">...
    </tbody>
  </table>
</div>
... #content-2-wide #main_bottom div#titleImageStrip.article
```

Styles (title-flat-...356\_.css:28)

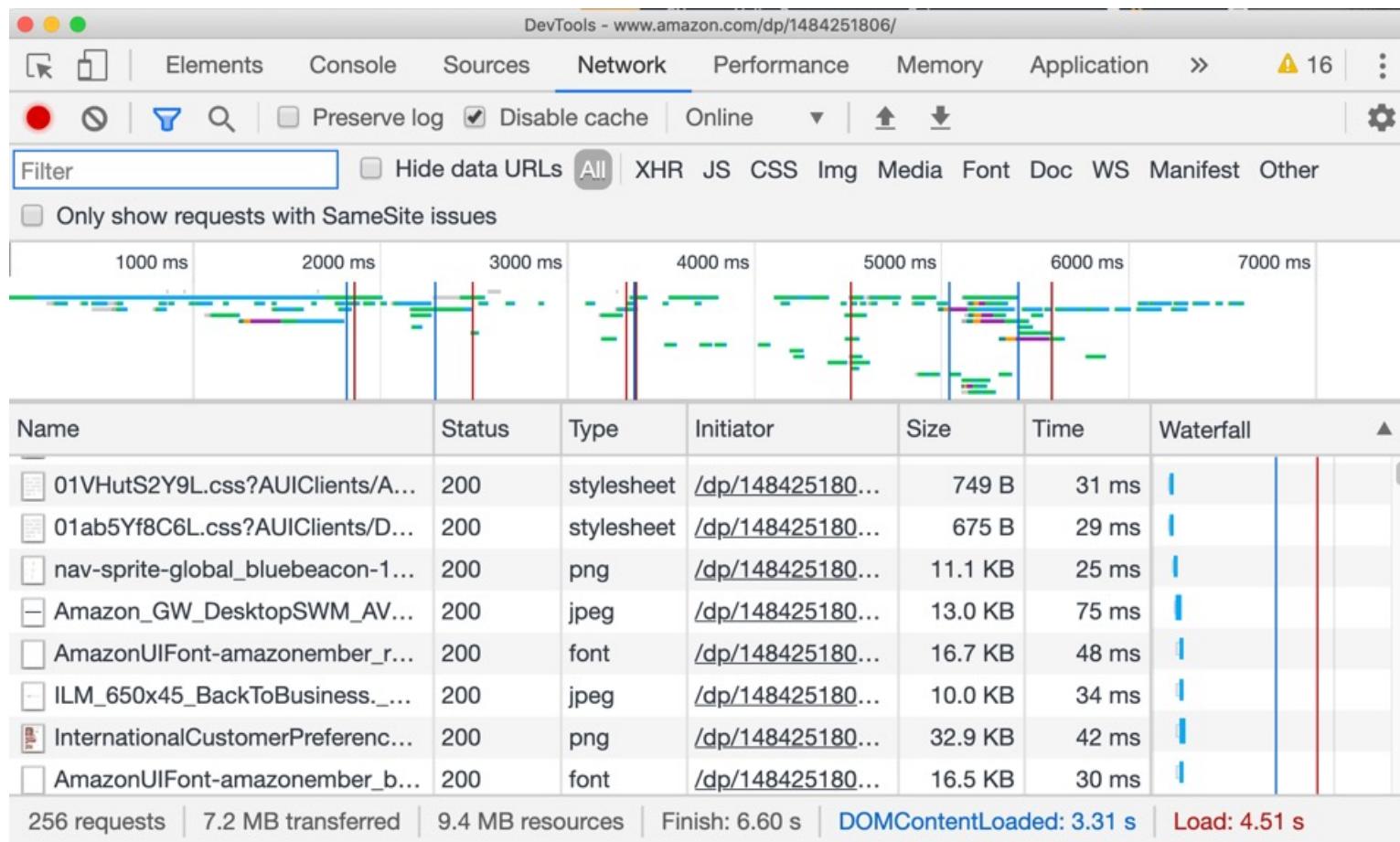
```
.article {
}
#content-2-wide #main .article,
#content-2-wide .main .article {
  border-top: 1px solid #CCC;
}
.article {
  border-top: none;
  padding: 20px 21px 24px 20px;
}
.article,
.article.on-tv,
.aux-content-widget-2,
.aux-content-widget-3,
.aux-content-widget-4,
.mini-article {
  background: none;
  border-radius: 0px;
  border: none;
  box-shadow: none;
  margin: 0;
  padding: 0;
```

# Debugging HTML and CSS

The screenshot shows the DevTools interface with the 'Elements' tab selected. The left sidebar displays the DOM tree for the URL [www.imdb.com/title/tt0096697/?ref\\_=fn\\_al\\_tt\\_1](http://www.imdb.com/title/tt0096697/?ref_=fn_al_tt_1). The 'Computed' tab in the top right is active, showing a detailed breakdown of the CSS properties for the selected element. A visual representation of the element's bounding box is shown with nested boxes in orange, yellow, and green, indicating margin, border, and padding respectively. The 'Computed' tab lists the following properties:

| Property              | Value            |
|-----------------------|------------------|
| background-attachment | scroll           |
| background-clip       | border-box       |
| background-color      | rgba(0, 0, 0, 0) |
| background-image      | none             |
| background-origin     | padding-box      |
| background-position-x | 0%               |
| background-position-y | 0%               |
| background-repeat-x   |                  |
| background-repeat-y   |                  |
| background-size       | auto             |

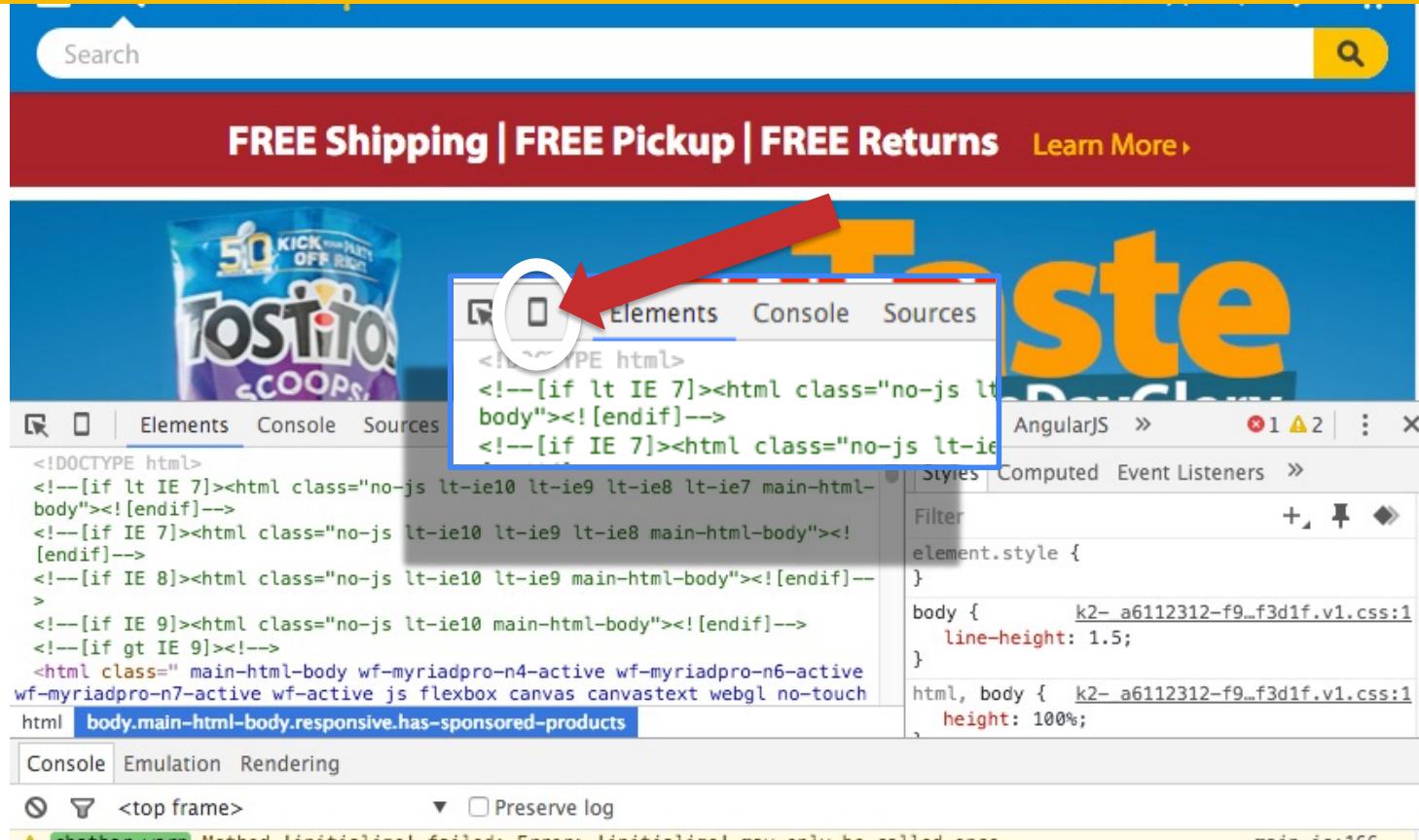
# Working with CSS in the Computed tab



# Examine traffic on the Network tab

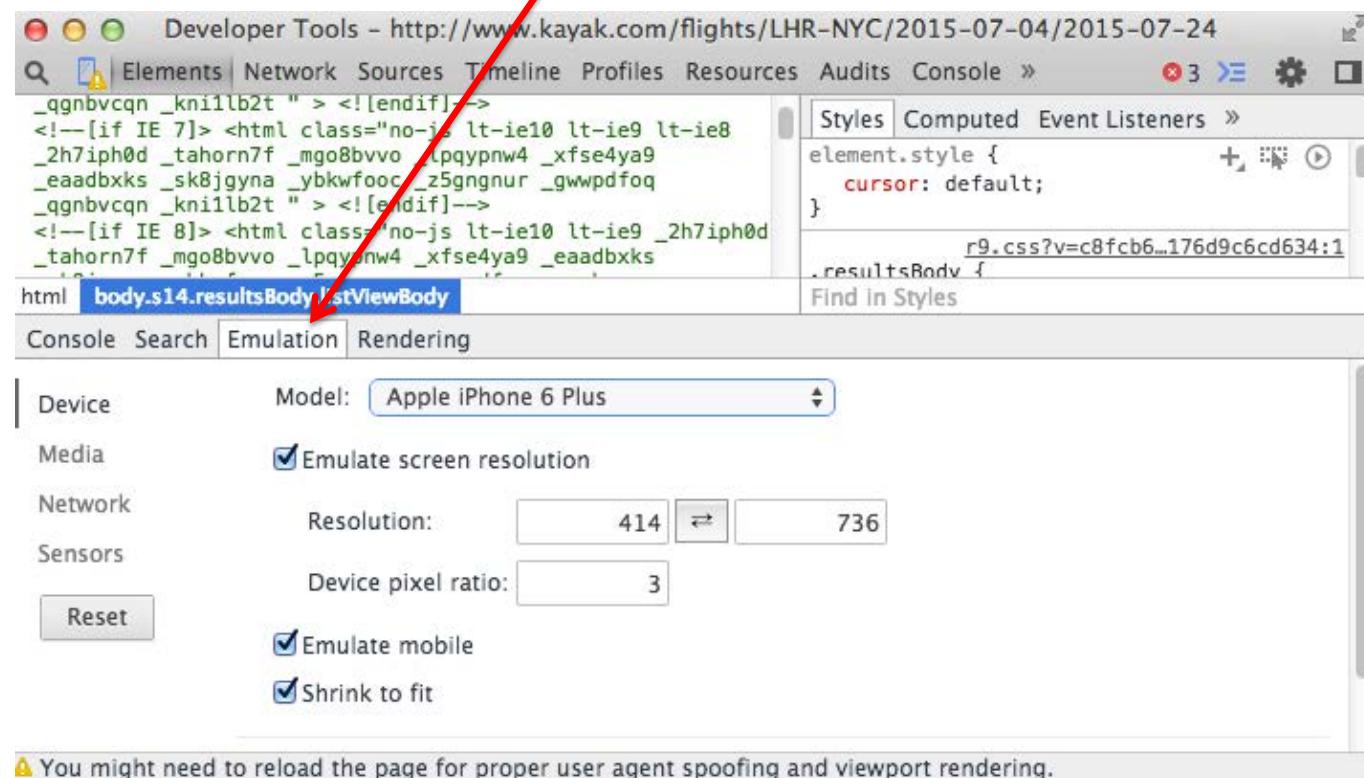
Shows each resource requested, its HTTP method, result, type, and speed

# You can simulate mobile devices



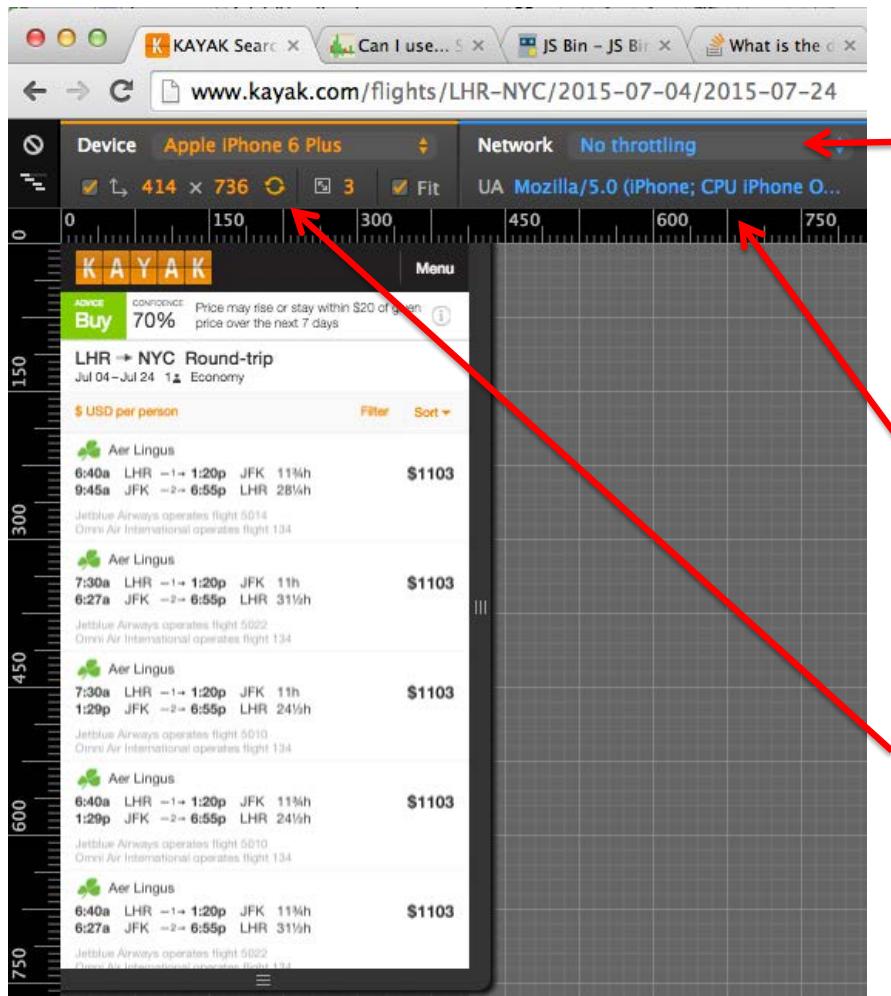
# Then pick your preferred device

Choose your device under  
the Emulation tab



# 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

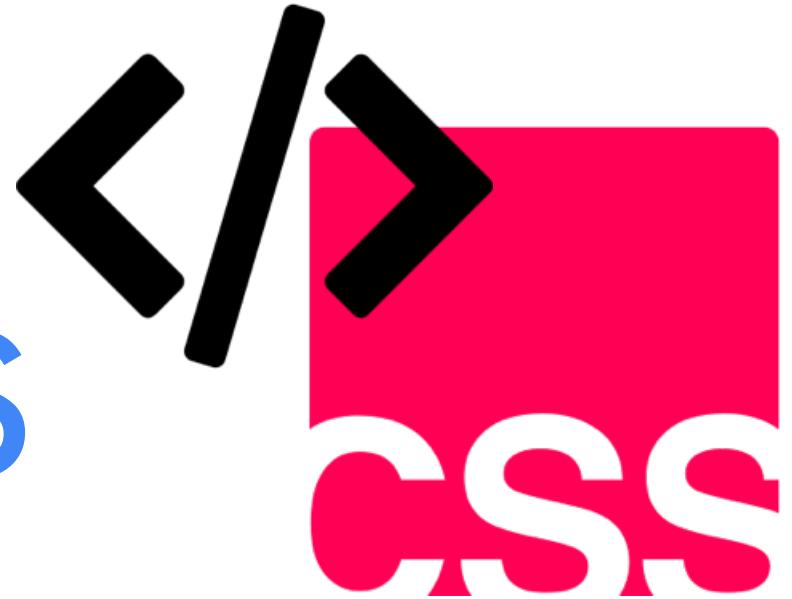
Choose  
your  
mobile  
browser  
agent

Rotate the  
device

## 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

# Effective CSS Styling



How to fine-tune your page's look

## 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

# 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!***

# 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;  
}
```

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

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



## ... Or on the page ...

```
<head>
<title>I &lt;3 Styles!</title>
<style>
.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>
```

... or for the entire site

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



A photograph of a waterfall cascading down a grassy hillside under a cloudy sky. The waterfall is on the left, and the hillside slopes down towards the right. In the background, there are more hills and mountains under a cloudy sky.

# 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

## 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")



## 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

## 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

# Overlapping properties

## All attributes are additive but if two are contradictory, who wins?

- Style > id > class > element
- A style overrides anything
- An element is overridden by everything else
- If you apply a class, that overrides an element
- An ID overrides a class.
- Last one to write wins

# This is !important

- You can prevent overriding with *!important*

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



**Use with caution! Makes  
it tough to debug.**

# CSS custom properties

... like variables in your CSS

## 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;  
}
```

## Solution: Create CSS variables!



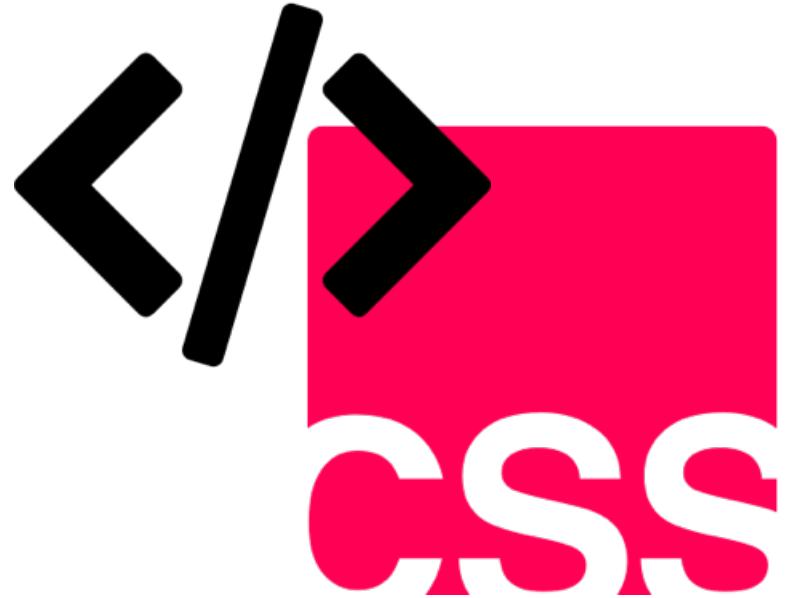
**Must begin with '--'**  
**Referenced with 'var(...)'**

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

## 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

# Semantic Grouping



So they can be positioned and styled as a unit

## 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

# We need groupings to specify areas of our pages.

W Phineas and Ferb - Wikipedia Rap

Secure https://en.wikipedia.org/wiki/Phineas\_a... Not logged in Talk Contributions Create account Log in Article Talk Read Edit View history Search Wikipedia

**Phineas and Ferb**

From Wikipedia, the free encyclopedia

This article is about the television show. For the soundtrack album, see *Phineas and Ferb (soundtrack)*. For the video game, see *Phineas and Ferb (video game)*.

**Phineas and Ferb** is an American animated musical comedy television series. Originally broadcast as a one-episode preview on August 17, 2007 and again previewed on September 28, 2007, the series officially premiered on February 1, 2008 on Disney Channel, and follows Phineas Flynn and his British stepbrother Ferb Fletcher<sup>[1]</sup> on summer vacation. Every day, the boys embark on some grand new project, which annoys their controlling sister, Candace, who frequently tries to reveal their shenanigans to her and Phineas' mother, Linda Flynn-Fletcher, and less frequently to Ferb's father, Lawrence Fletcher. The series follows a standard plot system:



|             |  |
|-------------|--|
| Genre       | Comedy<br>Musical  |
| Created by  | Dan Povenmire<br>Jeff "Swampy" Marsh   |
| Directed by | Dan Povenmire<br>Jeff "Swampy" Marsh<br>Zac Moncrief<br>Robert F. Hughes<br>Jay Lender<br>Sue Perrotto<br>Kim Roberson |

Wikipedia

Main page  
Contents  
Featured content  
Current events  
Random article  
Donate to Wikipedia  
Wikipedia store

Interaction  
Help  
About Wikipedia  
Community portal  
Recent changes  
Contact page

Tools  
What links here  
Related changes  
Upload file  
Special pages  
Permanent link  
Page information  
Wikidata item  
Cite this page

Print/export  
Create a book  
Download as PDF  
Printable version

In other projects

```
<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

# Elements come in two flavors: block and inline

## Block

- 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>`, `<form>`, `<input>`

# HTML5 semantic elements communicate *meaning*

 section

 nav

 article

 aside

 header

 footer

 main

 details

 summary

 figure

 hr

A photograph of a young woman with dark hair and glasses, wearing a white t-shirt, sitting at a desk and looking intently at a laptop screen. She has her hand to her chin in a thoughtful pose. A large red thought bubble originates from her head, containing the text "But why use semantic elements?".

But why  
use  
semantic  
elements?

## Quick quiz

**<p>** tags hold ...

... a generic paragraph of text

**<li>** tags hold ...

... list elements

**<img>** tags hold ...

... pictures, photos

**<div>** tags hold ...

... uh ... ummm ...



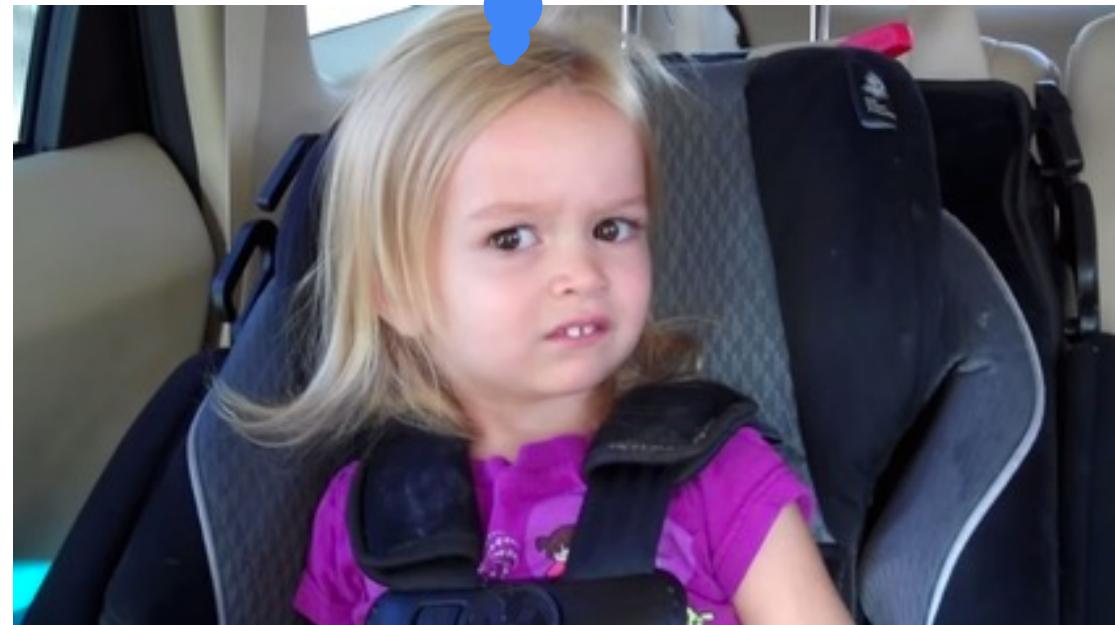
# Semantic elements add meaning to HTML

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

## 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?



## A section is for grouping general things in your document

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

# 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

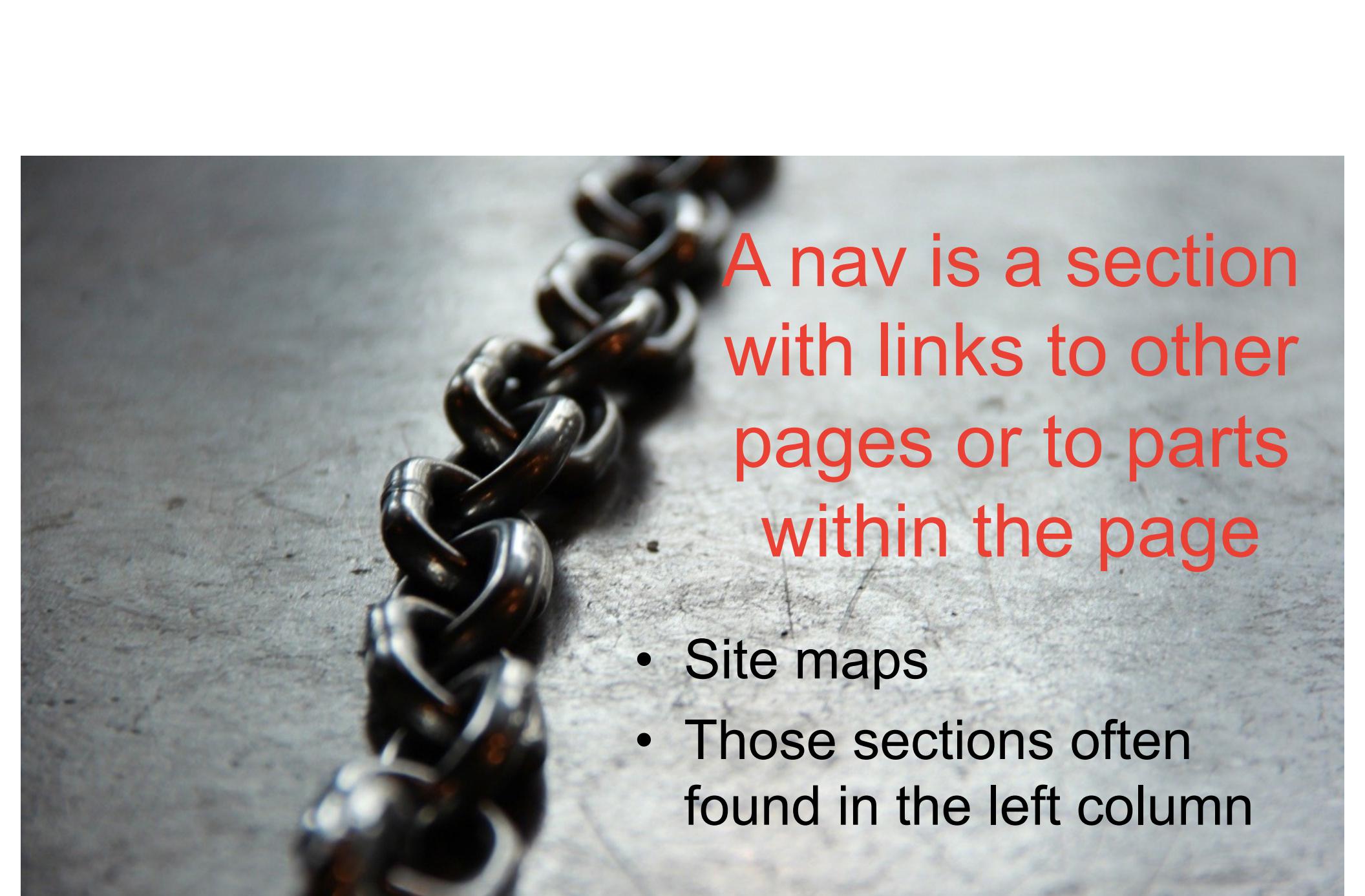
# Articles are for content that could be syndicated

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



# 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  |



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

May be  
a nav

The screenshot shows the eBay homepage with several sections highlighted by red arrows:

- A red arrow points from the left side towards the top navigation bar, which includes links for home, my eBay, site map, sign in, Browse, Sell, Services, Search, Help, and Community.
- A red arrow points from the left side towards the "Specialty Sites" section, which lists eBay Motors, eBay Premier (NEW!), Services for Business (NEW!), and Half.com (an eBay company).
- A large red circle highlights the "Categories" section, which lists numerous categories such as Antiques & Art, Books, Movies, Music, Business (Office & Industrial), Clothing & Accessories, Coins, Stamps, Collectibles, Computers, Network, IT, Dolls & Bears, Home & Garden, Jewelry, Gems, Watches, Photo, Electronics, Pottery & Glass, Real Estate, Sports (Memorabilia, Goods), Tickets, Travel, Toys, Hobbies & Crafts, Everything Else, and all categories... Below this section, it says Over 5 million items for sale!
- A red arrow points from the right side towards the "Hot Picks" section, which features a "Golf" banner with tournaments, tickets, tee times, and more.
- A red circle highlights the "Local Trading" section, which includes links for Appliances, Sporting Goods, Furniture, and more, along with a dropdown menu for picking a region and a "Go!" button.
- A red circle highlights the "Browse by Themes" section, which includes links for Sports Stars, Home Theater, Travel, and more, along with a dropdown menu for All Themes and a "Go!" button.
- A red circle highlights the "Featured Items" section, which displays a list of items like New - Designer Style Leather Umbrella Bag/NR, #1 Diet Pill Lose 98 Lbs By Summer Guaranteed!, #1 Diet Pill Lose 88lbs By June See Proof!, Lose 88 Pounds Or 8 Sizes & Vission On Us!, #1 Diet Pill Lose 88 Lbs By Summer Guaranteed, #1 Diet Pill Lose 50, 100, 200 Lbs. Guaranteed, and all featured items... To the right of this section is a "Don't Miss..." box featuring a woman named Rosie and a globe icon, with a "Charity" link below it.
- A red arrow points from the right side towards the "Gallery Items" section, which includes a link for all items... Below this section is the "Spotlight's On..." section, which features four items: Cruise & Golf Excursion, Golf Equipment, a holt.com BUY NOW! offer for new buyers getting \$5 off \$10!, and Discover Electronic Checks.

Probably  
not a nav

A header contains the stuff at the top of a section

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



- 
- A woman with long dark hair in a braid, wearing a light blue button-down shirt, is pointing her index fingers downwards with both hands. This gesture typically indicates where to look for additional information or where something is located.
- About the author
  - Company data
  - Links
  - Date it was written
  - Copyright notice

A footer contains the stuff at the bottom of a section

# <main>

- Newest of the 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>
```

<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
```

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

**Do this ...**

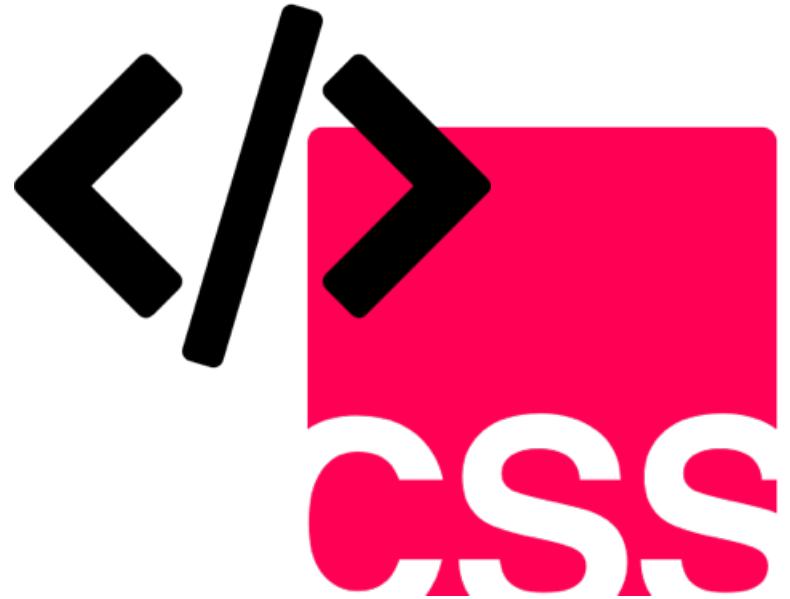
**... not this**

```
<div class="figure">  
    
  <p>  
    Perry (Agent P)  
  </p>  
</div>
```

## 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

# How to Position with CSS



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

## 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 is tougher than you'd think - use flexbox
- position: absolute/relative/fixed/static

**Zen Garden**

A demonstration of what can be accomplished visually through CSS-based design. Select any style sheet from the list to load it into this page.

*The Road to Enlightenment*

Littering a dark and dreary road lay the past relics of browser-specific tags, incompatible DOMs, and broken CSS support.

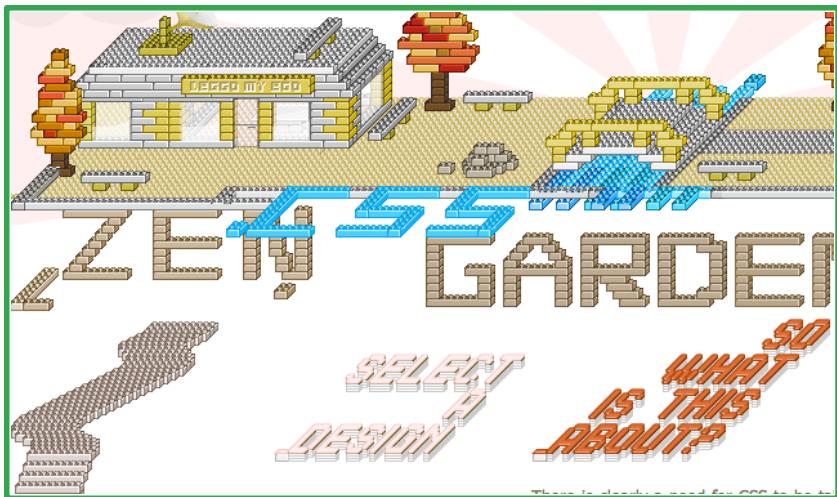
Today, we must clear the mind of past practices. Web enlightenment has been achieved thanks to the tireless efforts of folk like the W3C, WaSP and the major browser creators.

The css Zen Garden invites you to relax and meditate on the important lessons of

[Download the sample html file](#) and [css file](#)

**select a design:**

- [Under the Sea! by Eric Stoltz](#)
- [Make 'em Proud by Michael McAgan and Scotty Reifsnyder](#)
- [Orchid Beauty by Kevin Addison](#)
- [Oceanscape by Justin Gray](#)
- [CSS Co., Ltd. by Benjamin Klemm](#)



**csszen garden**

A DEMONSTRATION OF WHAT CAN BE ACCOMPLISHED VISUALLY THROUGH CSS-BASED DESIGN. SELECT ANY STYLE SHEET FROM THE LIST TO LOAD IT INTO THIS PAGE.

[DOWNLOAD THE SAMPLE HTML FILE](#) | [DOWNLOAD THE SAMPLE CSS FILE](#)

**THE BEAUTY OF CSS DESIGN**

**The Road to Enlightenment**

Littering a dark and dreary road lay the past relics of browser-specific tags, incompatible DOMs, and broken CSS support.

Today, we must clear the mind of past practices. Web enlightenment has been achieved thanks to the tireless efforts of folk like the W3C, WaSP and the major browser creators.

The css Zen Garden invites you to relax and meditate on the important lessons of the masters. Begin to see with

**SELECT A DESIGN**

**UNDER THE SEA**  
by [Eric Stoltz](#)

**MAKE 'EM PROUD**  
by [Michael McAgan And Scotty Reifsnyder](#)

**ORCHID BEAUTY**  
by [Kevin Addison](#)

**OCEANSCAPE**  
by [Justin Gray](#)

**CSS CO., LTD.**  
by [Benjamin Klemm](#)

**SAKURA**  
by [Tatsuya Uchida](#)

**KYOTO FOREST**  
by [John Politowski](#)

**THE ROAD TO ENLIGHTENMENT**

Littering a dark and dreary road lay the past relics of browser-specific tags, incompatible DOMs, and broken CSS support.

Today, we must clear the mind of past practices. Web enlightenment has been achieved thanks to the tireless efforts of folk like the W3C, WaSP and the major browser creators.

The css Zen Garden invites you to relax and meditate on the important lessons of the masters. Begin to see with clarity. Learn to use the (yet to come) tools of the web in a new and invigorating fashion. Become

# Sizing things on a page

Laying out sections of a page

# "display" lays out me or the things inside me

How do I get laid out in relation to the things around me?

- block
- inline
- inline-block

How do my children get laid out in relation to the other things inside me?

- flex
- grid
- table
- list-item



There are some more values

## Inline

- Flows along with the text around it
- No concept of width or height
- Ignores the box model

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt. **This is some text inside a container. It is set out differently than its surroundings** Ut labore et dolore magna aliqua. Ac tortor vitae purus faucibus ornare suspendisse sed

## Block

- Creates a clean break before it and after it
- Allows you to set width and height
- Honors the box model

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt. **This is some text inside a container. It is set out differently than its surroundings** Ut labore et dolore magna aliqua. Ac tortor vitae purus faucibus ornare suspendisse sed

# The box model provides aesthetic space

## Margin

- Space outside the container between sections

## Padding

- Space inside the container between the container and the contents



# There are many units of measure

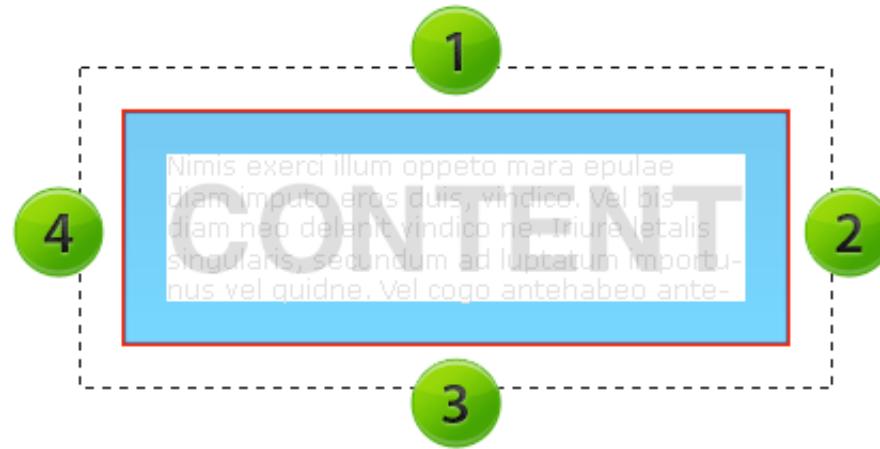
- **Pixels (px)**
  - Affected by screen resolution
  - Great for absolute control over layout
- **Ems (em)**
  - Proportion of the default font of the browser
  - Great for accessibility
- **Percentages (%)**
  - Proportion of the width or height of the container
- **Viewport size (vh and vw)**
  -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

## So, which do I use?

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

# You can have different box sides

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



Top, Right, Bottom, Left

`padding: 1px;`

All four

`padding: 1px 2px;`

top & bottom

left & right

`padding: 1px 2px 3px;`

top

left & right

bottom

`padding: 1px 2px 3px 4px;`

top

right

bottom

left

Using  
shorthands  
can save  
characters

# Centering

# Horizontal centering

`text-align: center;`

- only works on inline elements and not block elements

`margin: 0px auto;`

- For block elements and not inline.
- The auto means make it the same on both sides.

# Vertical centering

`vertical-align: top; /* middle, bottom */`

- 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.

# 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.**

# The position style

The position style has discrete values:

static

relative

absolute

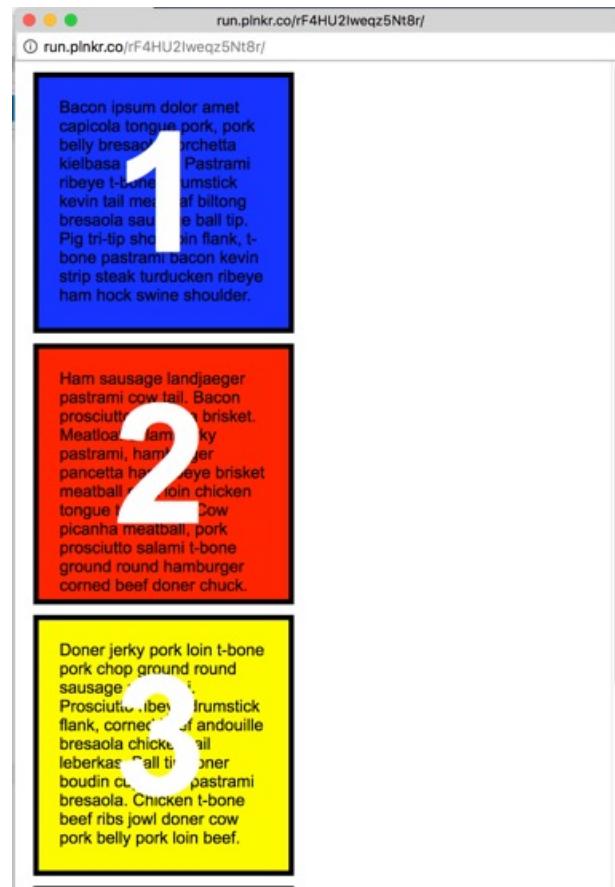
fixed

(and a few others)

- 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;
}
```

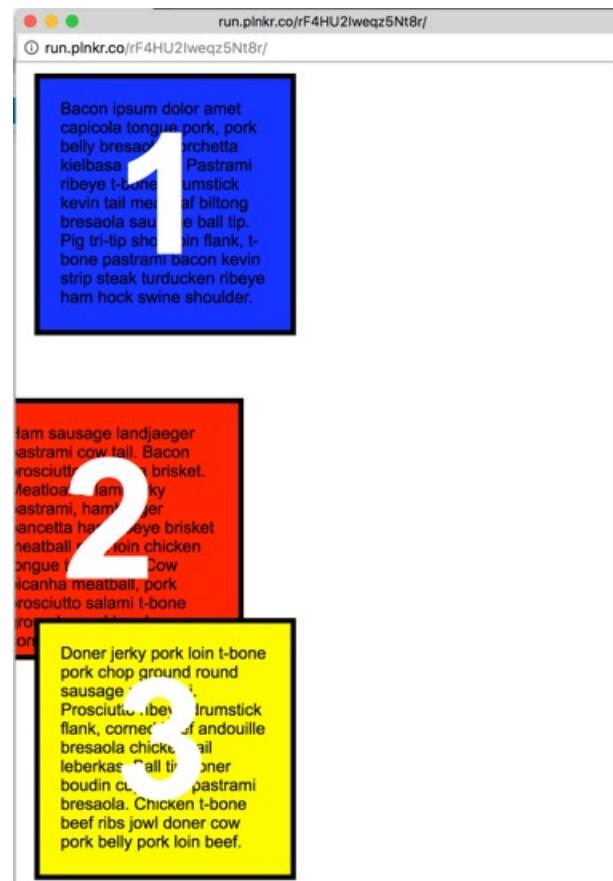
**static** (the default)



# 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

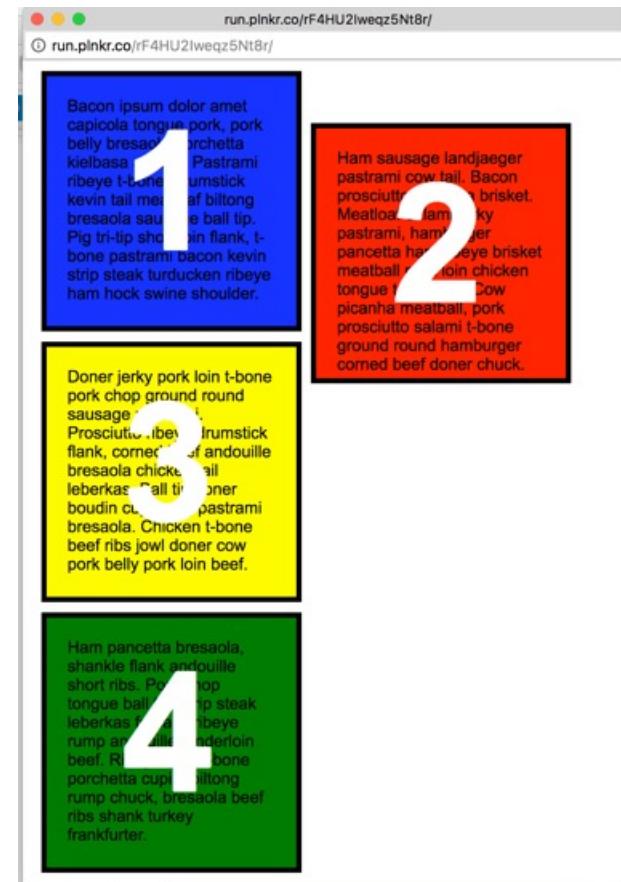
```
div:nth-child(2) {  
  position: relative;  
  top: 50px;  
  right: 50px;  
}
```



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

absolute

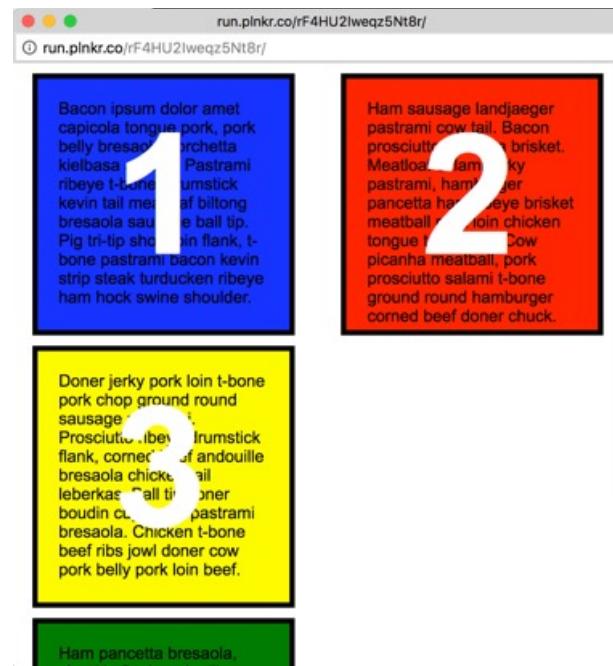
```
div:nth-child(2) {
  position: absolute;
  top: 50px;
  right: 50px;
}
```



fixed

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

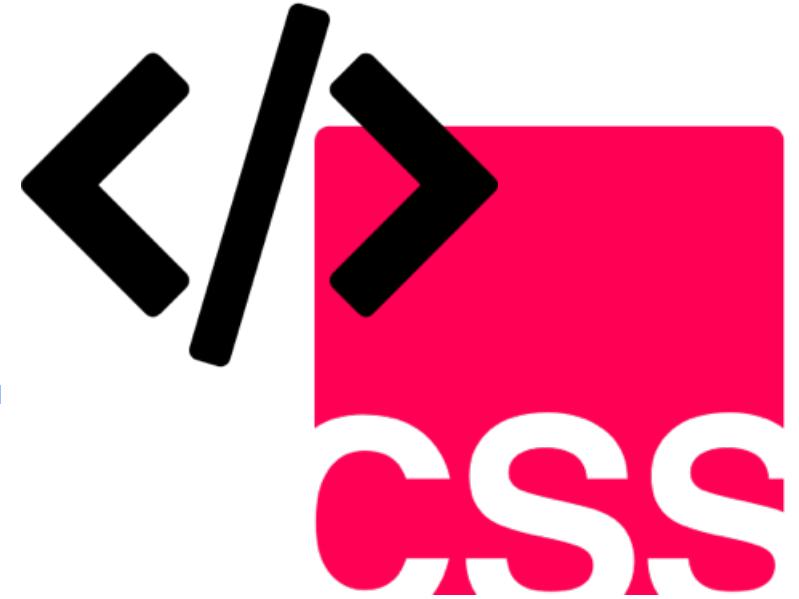
```
div:nth-child(2) {  
  position:fixed;  
  top: 0px;  
  right: 0px;  
}
```



## 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 is tougher than you'd think - use flexbox
- position: absolute/relative/fixed/static

# Page Layout Strategy



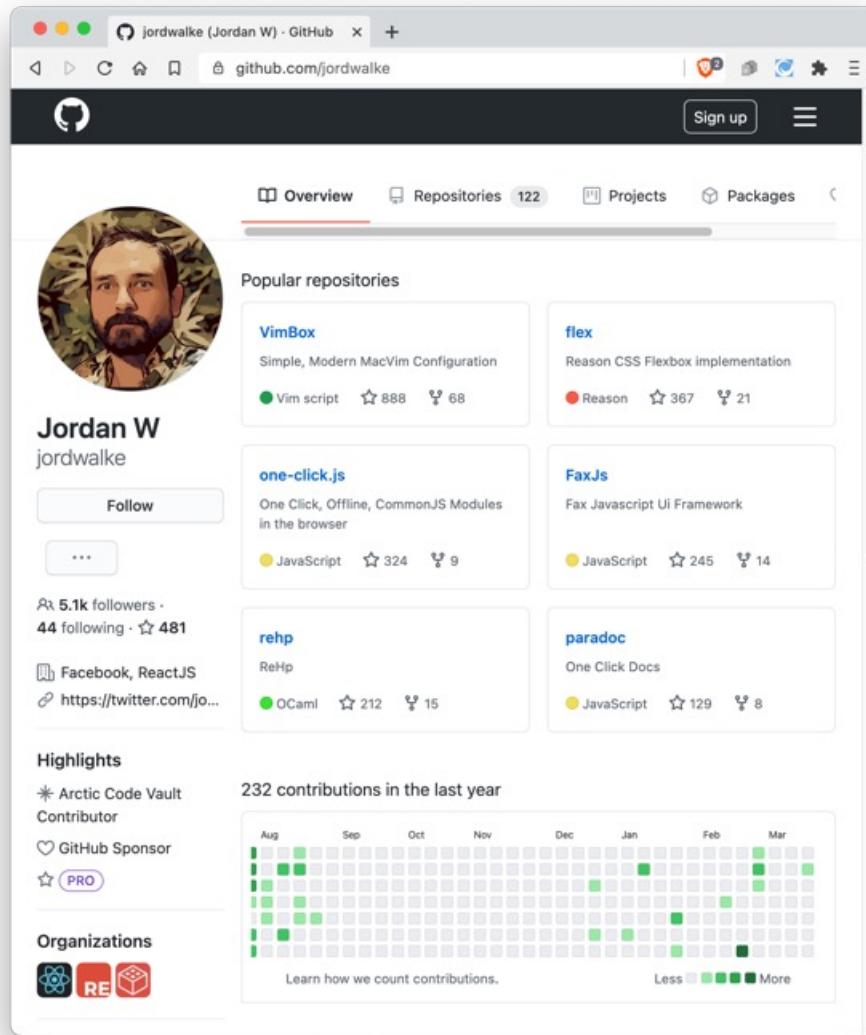
What are my options and which do I  
choose?

## 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?

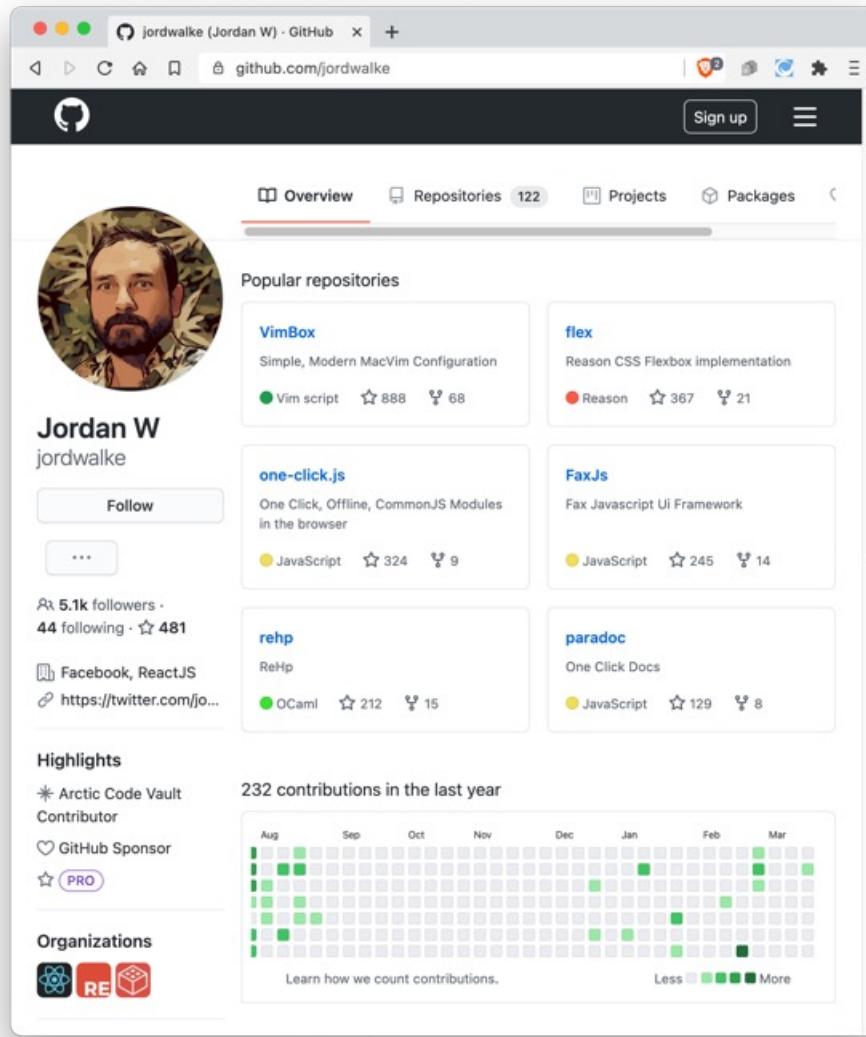
# How do you layout pages?

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



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

- Tables
- Absolute positioning
- Inline sections



## 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

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

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

# Floated Divs

Option 1

## 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

## 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

# When is the time to use float?

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



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

# **display: inline-block**

Option 2

## The good ...

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

## The bad ...

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

# The ugly ...

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

## Links

- [Nam ullamcorper](#)
- [commodo nibh](#)
- [tincidunt congue](#)
- [Vestibulum id](#)
- [Vivamus sed](#)

Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Suspendisse et dui dolor. Nam ac neque neque, sed mollis dolor. Etiam cursus nibh non elit lobortis id condimentum tortor porttitor.

## 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 lacinia. Donec a quam ut magna adipiscing molestie.

Ads



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

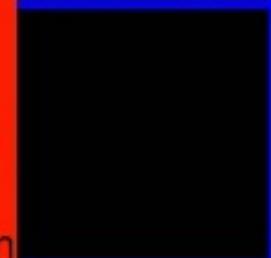
## Links

- [Nam ullamcorper](#)
- [commodo nibh](#)
- [tincidunt congue](#)
- [Vestibulum id](#)
- [Vivamus sed](#)

## 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 lacinia. Donec a  
quam ut magna adipiscing molestie.

## Ads



## To fix it ...

```
section {  
  display: inline-block;  
  width: 1px;  
}  
section div {  
  vertical-align: top;  
  margin: 0px -4px;  
}
```

A very high-level intro to  
flexbox and grid

# 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

# Grid

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

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

So ... which one do I learn?

|                               |   |
|-------------------------------|---|
| <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<br>(ie. rows <u>or</u> columns) |
| <b>display:grid;</b>          | 2d layouts (ie. rows <u>and</u> columns)                                  |

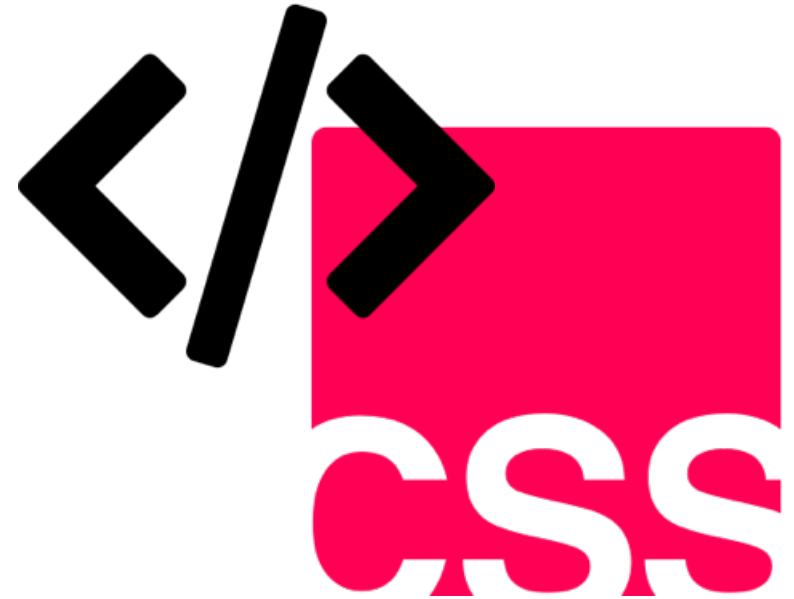
## Indeterminate number of items

- 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.

## 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?

# Layouts with Flexbox



The deep dive into CSS flexbox

## tl;dr

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

# Flex boxes



**Cary**  
@caryhartline

.@jensimmons it's difficult to te  
too complicated or if my mi  
mindset.



**Curtis**  
 @\_CurtisR

#flexbox

so difficult when it comes to the



**AlHasan Husni**  
 @AlHasanps

Flexbox is awesome  
concepts that  
interactive...

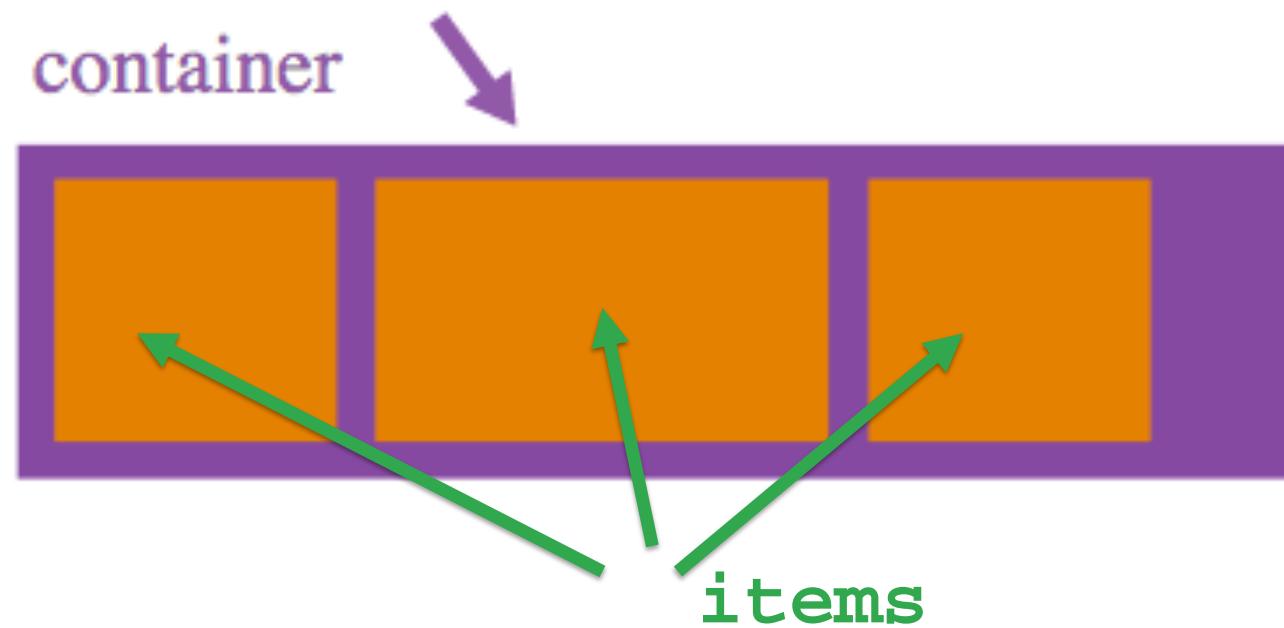


**Philip Roberts**  
 @philip\_roberts

I know flexbox is powerful and awesome; but  
boy does it make me feel like a total idiot  
everytime I try and use it.



flex involves a container and contents



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

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

## Mark the container as a flex-box

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

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



The container has axes

Most of us would call these

- horizontal
- vertical

## The container lays out the items

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



# Two ways to think about flexbox ...

Are we going to wrap or not?

## Two ways to think about flex

### We need it to not wrap

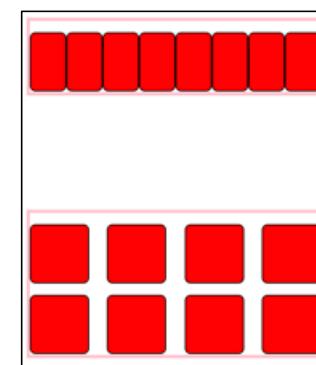
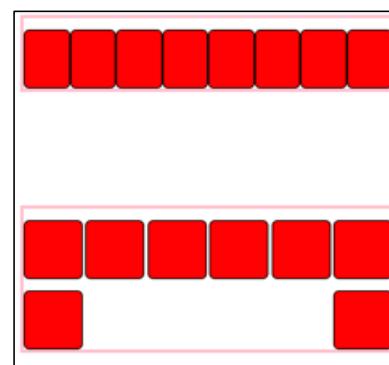
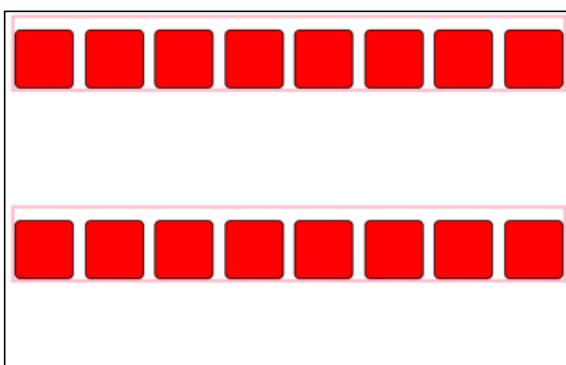
- For when you have a small and fixed number
- You want the space allocated to each to flex
- You'll use flex-basis, flex-shrink, and flex-grow.

### 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

# 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.



← no-wrap

← wrap

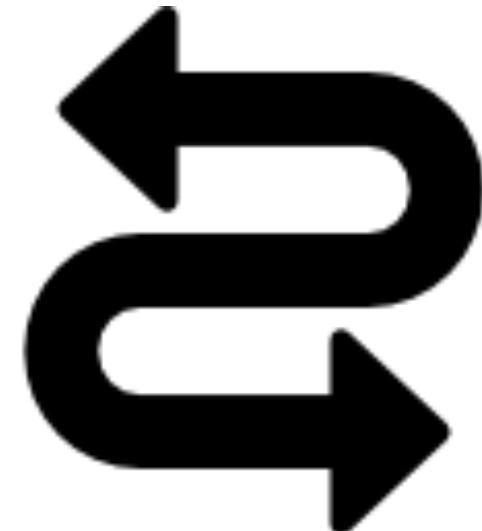


**Note: As the width decreases, the no-wrap row has to narrow the boxes to all fit. While with the wrap row, they can stay their favorite size.**

**Understanding flexbox is easier if  
you separate the two cases**



**no-wrap**



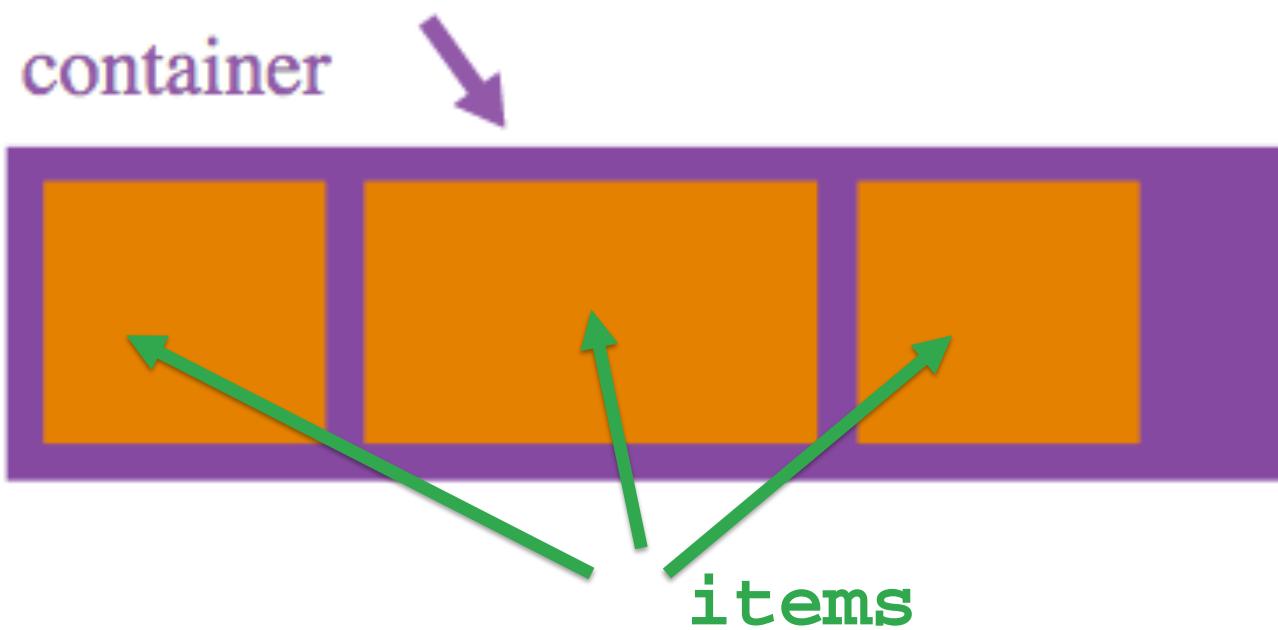
**wrap**

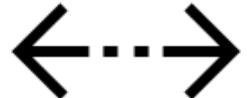


# Flexbox - no-wrap



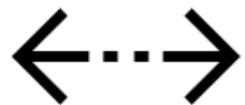
## flex items have sizes





## Sizes of the items

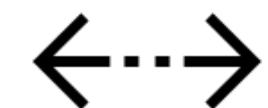
- **flex-basis** = the item's "favorite" size along the flex-direction.
  - in px, em, %, etc.
- If the viewport is increased from flex-basis ...
- **flex-grow** = by how much it grows
  - unitless - a relative number
- if the viewport is decreased from flex-basis ...
- **flex-shrink**= by how much it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



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.



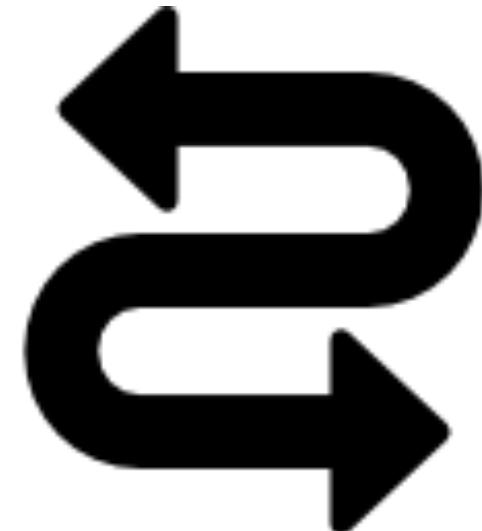
order

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

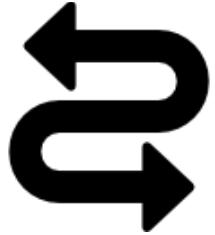
**Understanding flexbox is easier if  
you separate the two cases**



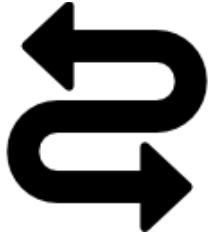
**no-wrap**



**wrap**

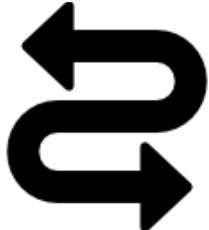


# Wrapping with flex



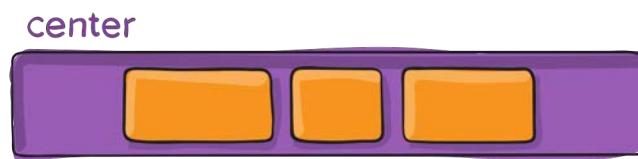
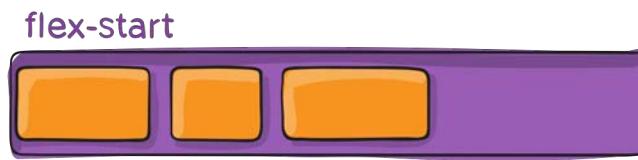
## flex-wrap tells them to wrap when needed

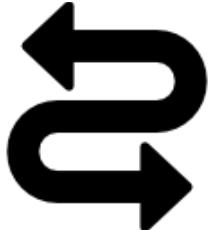
```
#container {  
    flex-wrap: wrap;  
}
```



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

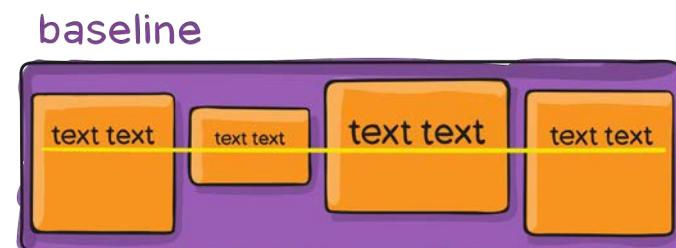
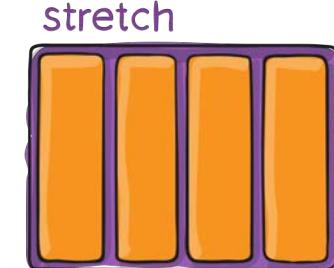
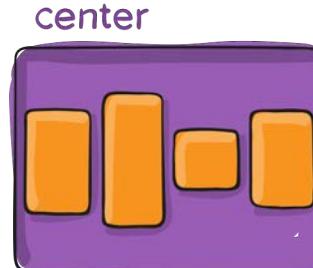
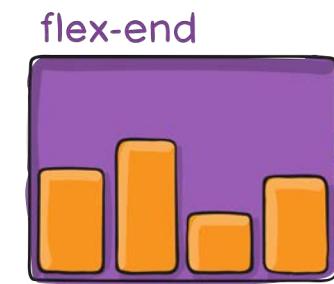
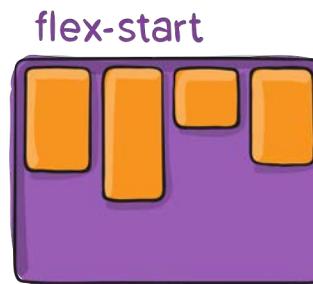
And the  
container  
controls the  
spacing





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

And the  
container  
controls the  
spacing



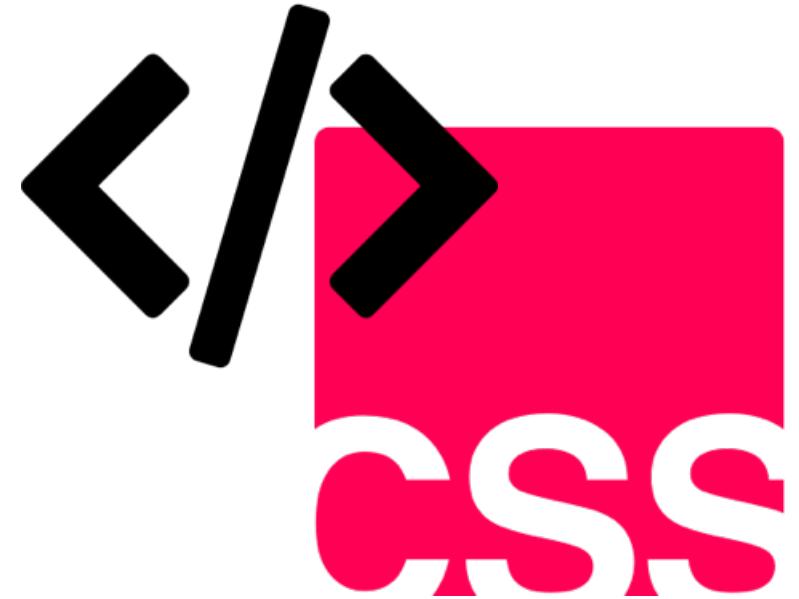
## Bonus! How to center anything

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

## tl;dr

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

# Layouts with Grid



The deep dive into CSS grids

## 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.

## 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*

## 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).

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

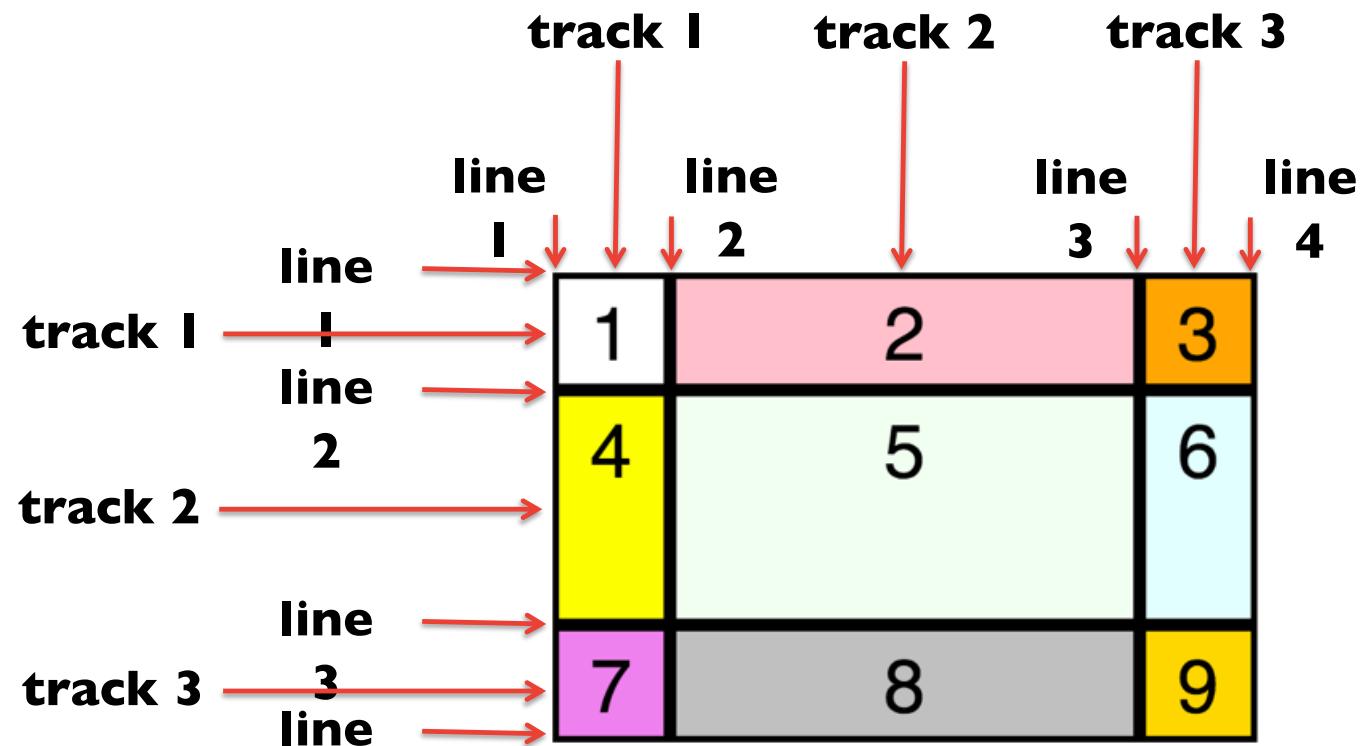
- "Free space" = what is left over after painting.
- They're distributed according to the proportional numbers of the columns.
- Why not just use %?
- Because fr takes into account grid gaps. % does not. If there are no grid gaps, they have the same effect.

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

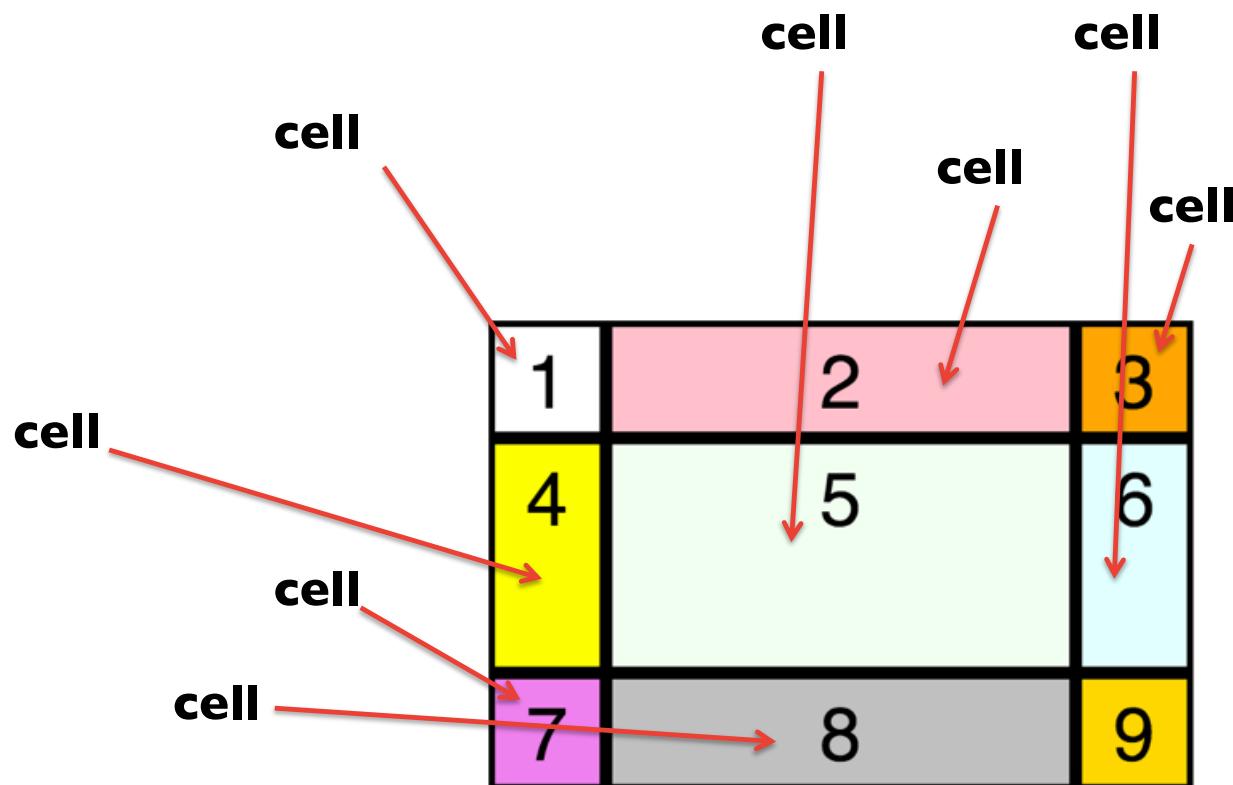
- A grid will place all of its direct children
- A grid will only place its direct children

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

grid lines  
separate  
the grid  
tracks



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



# 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.

```

.grid {
  display: grid;
  grid-template-columns: 50px 100px 50px;
  grid-template-rows: 50px auto auto;
}

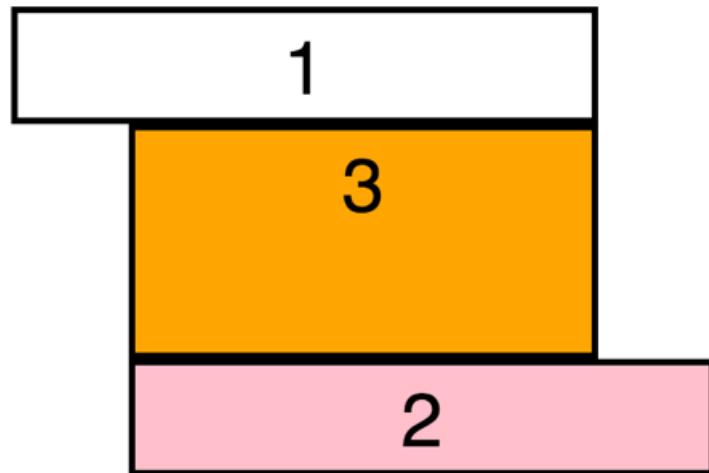
#one {
  grid-column: 1/3;
  grid-row: 1/2;
}

#two {
  grid-column: 2/4;
  grid-row: 3/4;
}

#three {
  grid-column: 2/3;
  grid-row: 2/3;
}

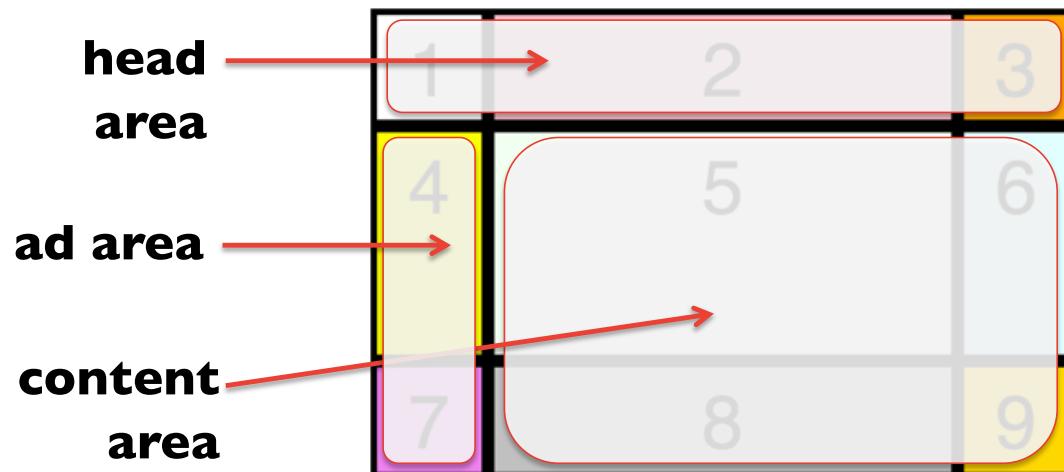
```

To assign to cells, place between lines



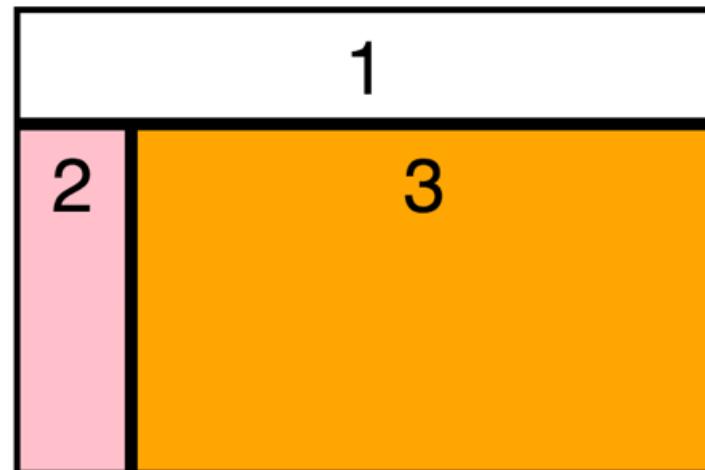
```
.grid {  
  display: grid;  
  grid-template-columns: 50px 100px 50px;  
  grid-template-rows: 50px auto auto;  
  grid-template-areas:  
    'head head head'  
    'ad content content'  
    'ad content content'  
}
```

**grid areas**  
are  
optional  
groups of  
cells



```
.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;  
}
```

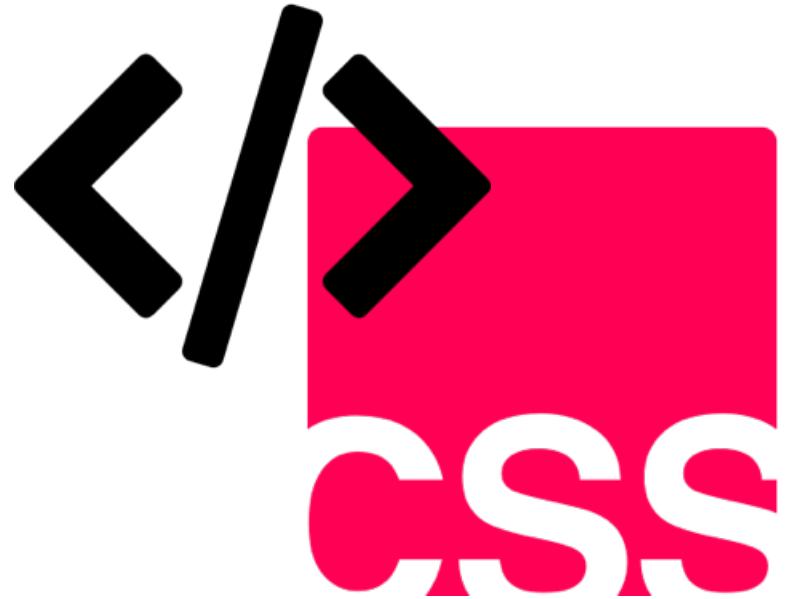
To  
assign to  
areas ...



## 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.

How do I  
make this  
layout?



Subtitle

There are  
different ways  
to create the  
same layout.

**Warning!** If  
you choose  
wrong, the  
CSS Police  
will come and  
arrest you.



Just  
kidding!

If you can make it  
work, awesome!  
There is no wrong  
way to do it. If it  
works, it works!



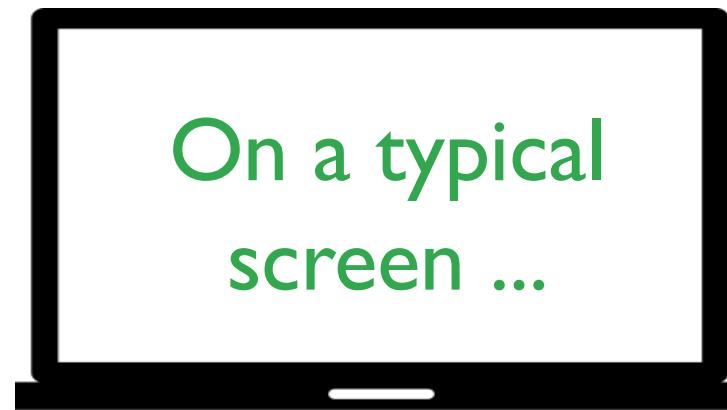
Most devs get good at one technique or the other. Once they do, they make it work for every situation. Is that good?

Or is it better to learn both and choose the best tool for the job?



# Let's play a game!

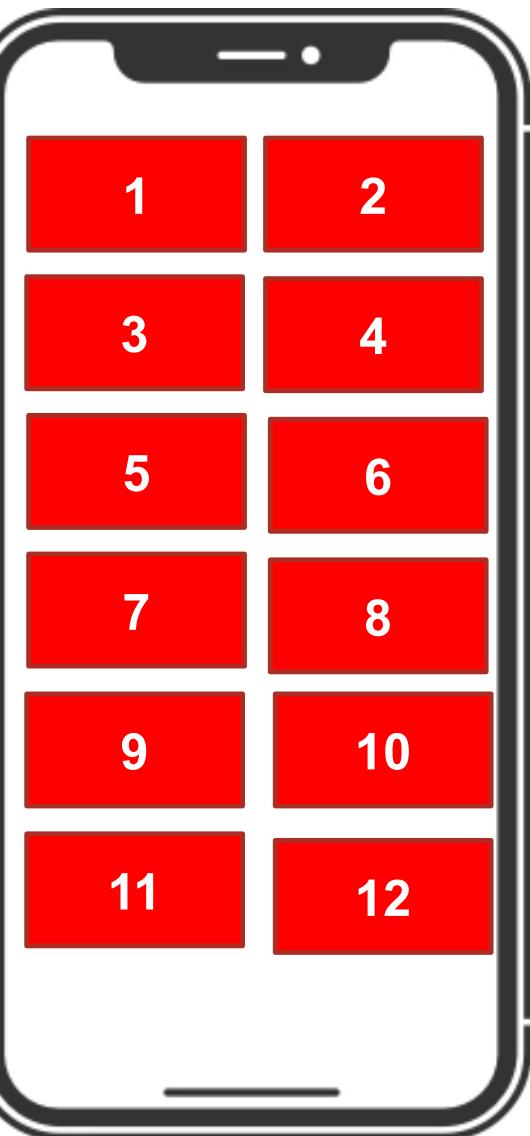
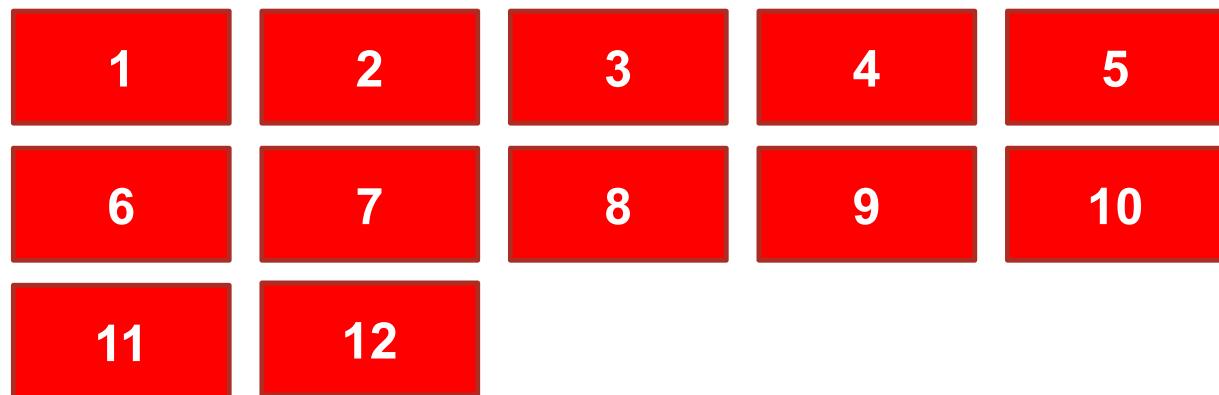
Over the next few pages, you'll see a layout.

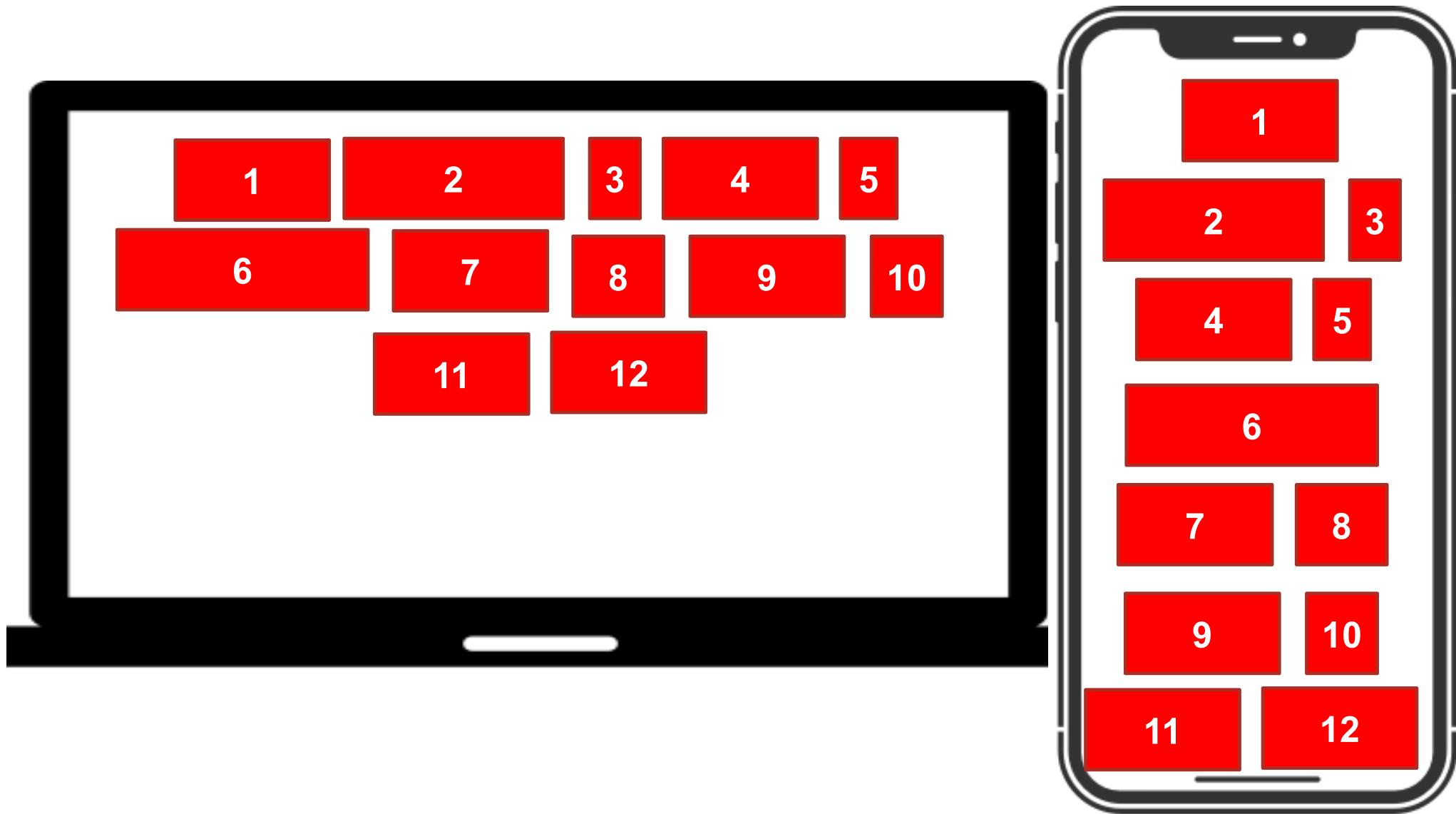


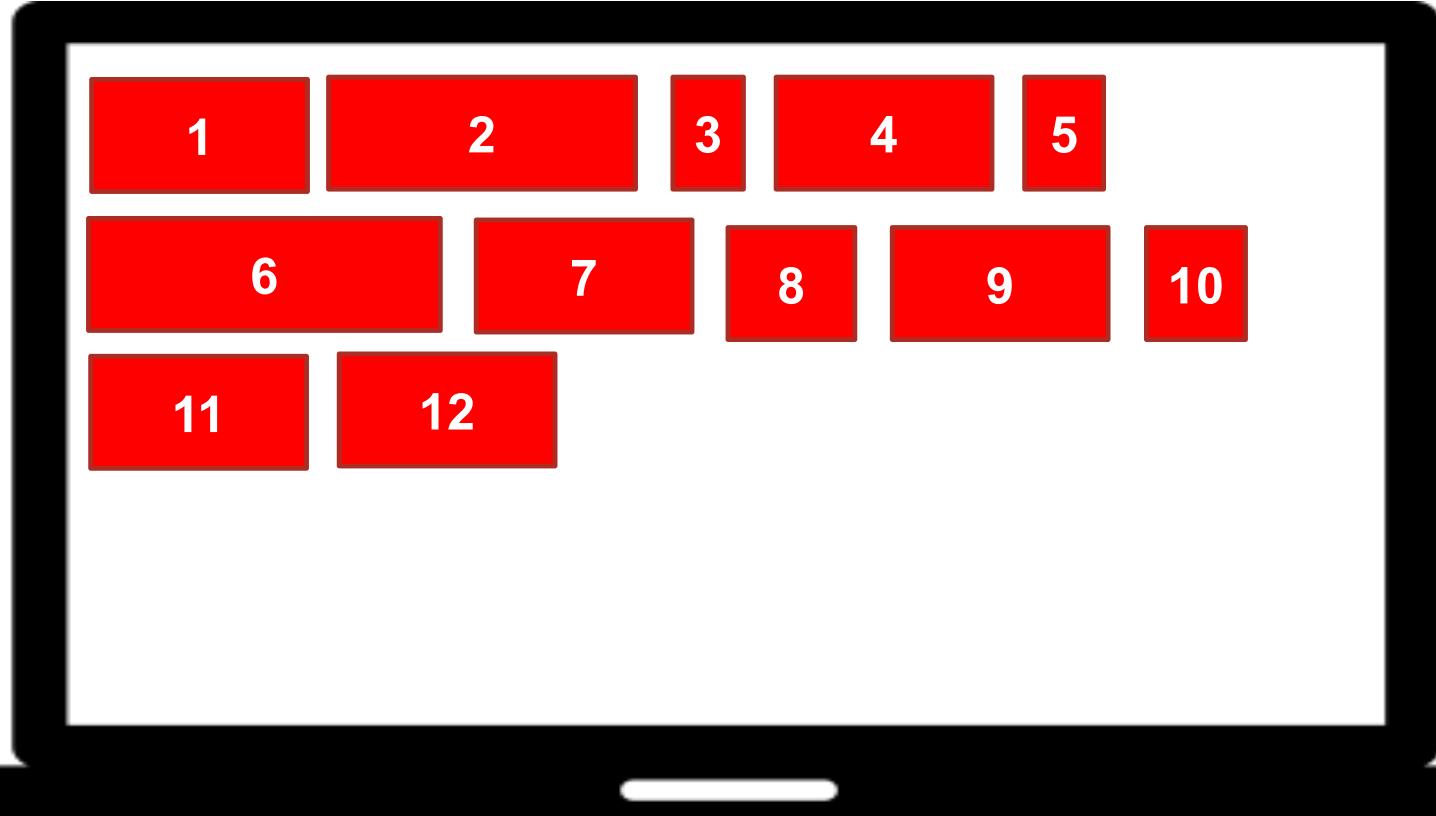
Your job will be to say

1. which technique would be best
2. your strategy to carry it out

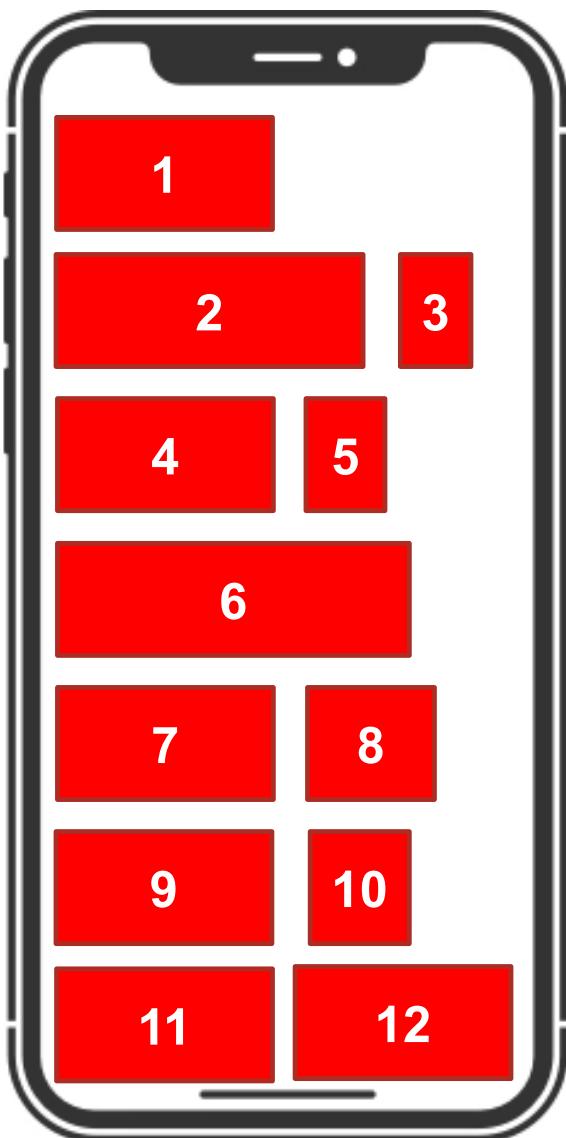
Ready?  
Let's go!



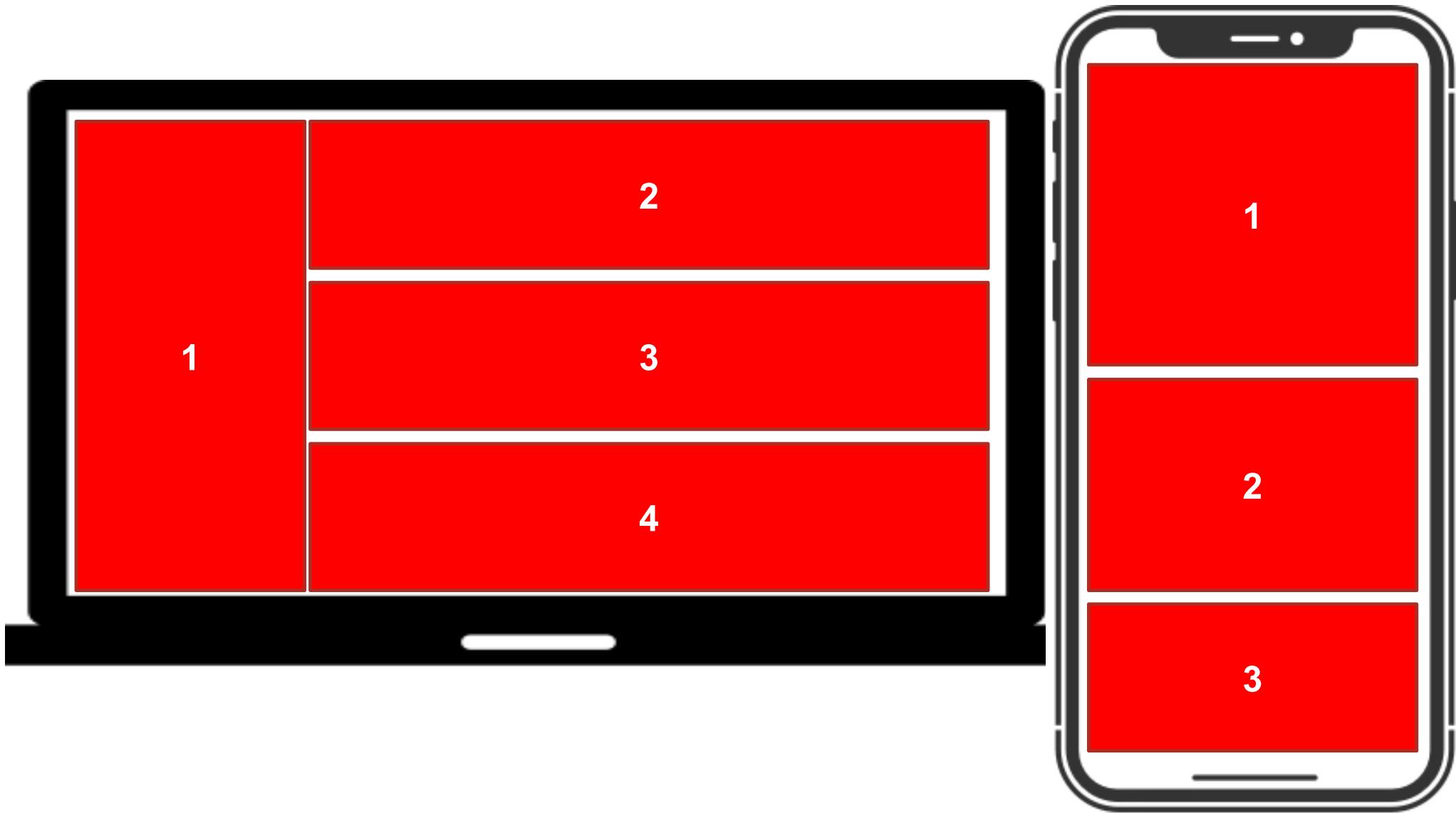


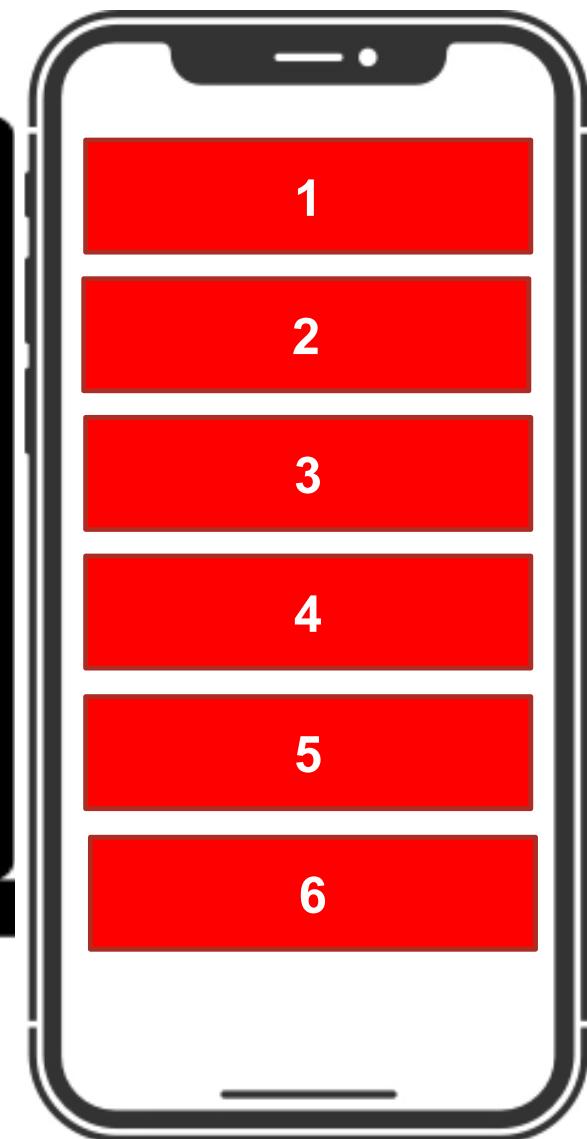
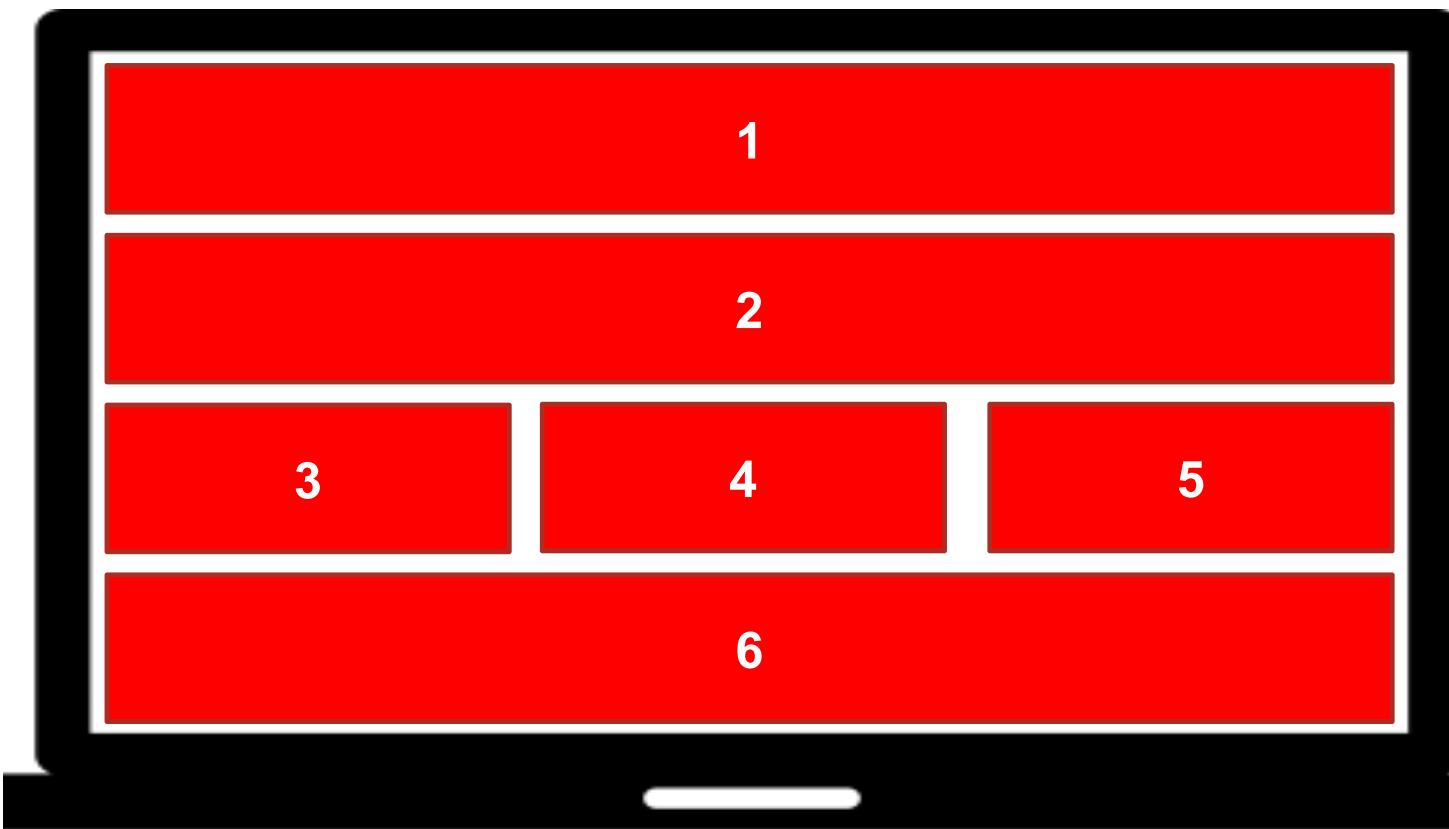


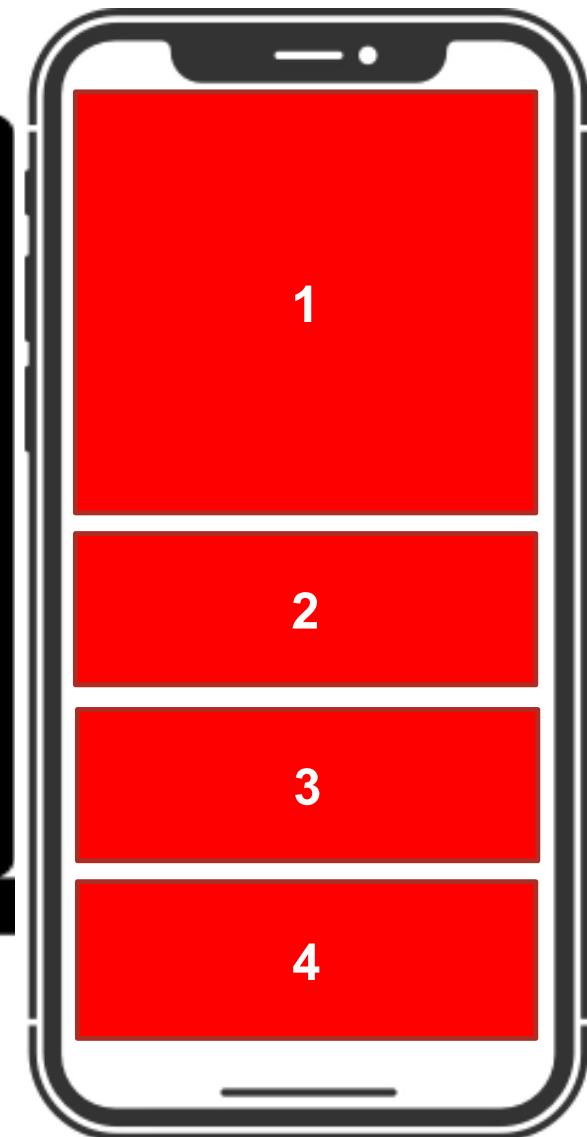
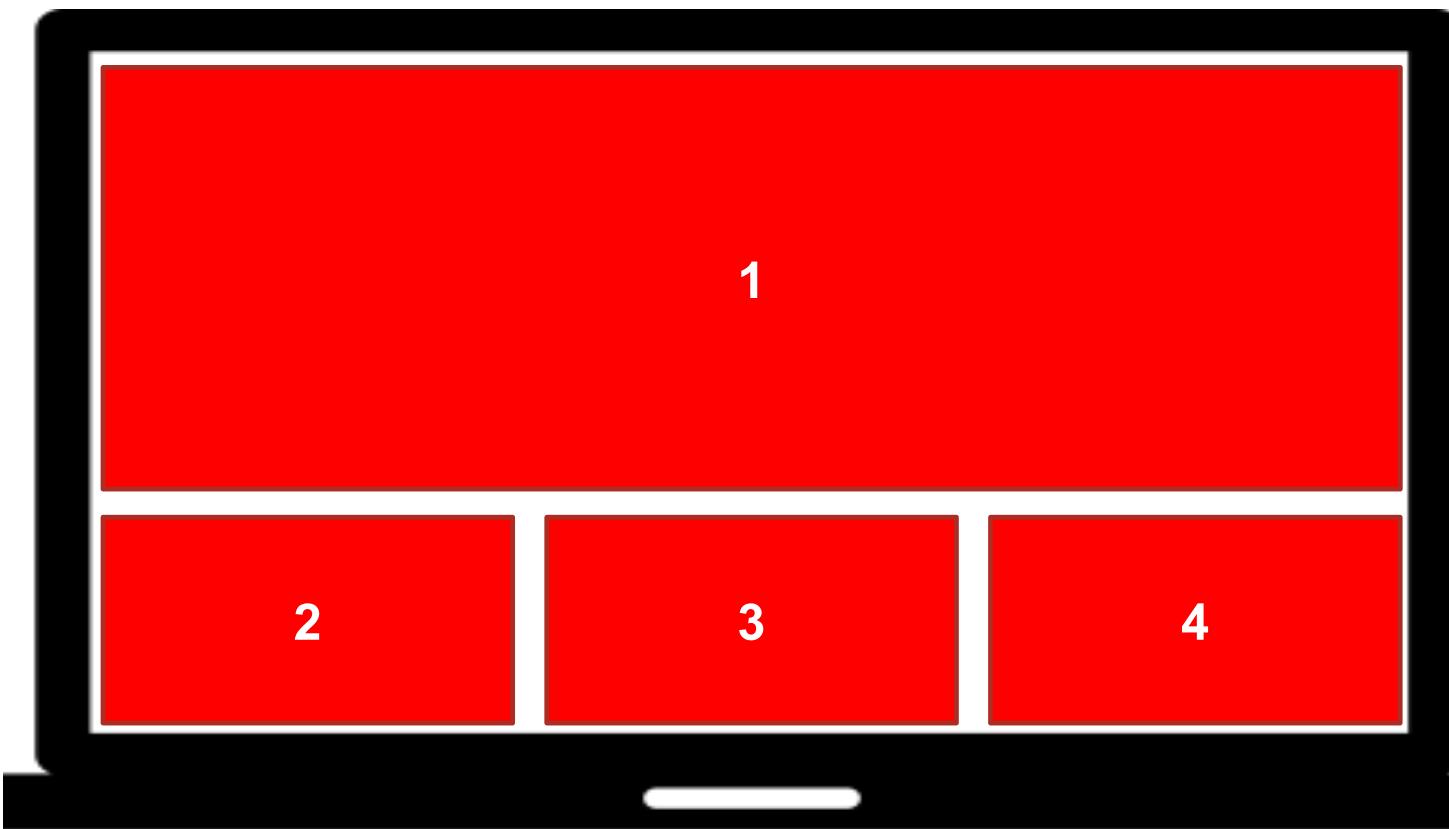
1 2 3 4 5  
6 7 8 9 10  
11 12

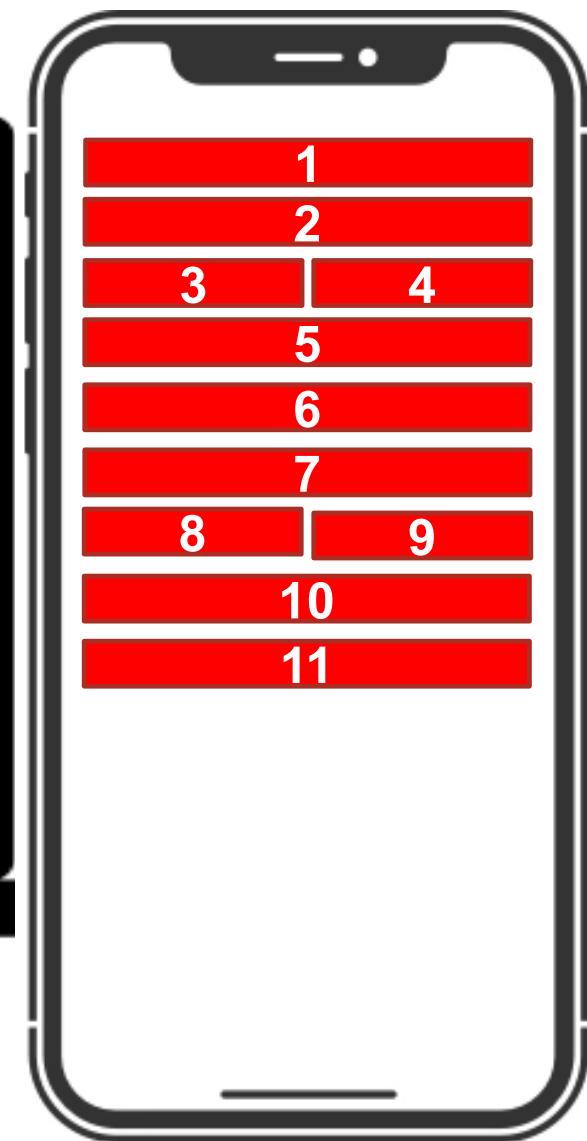
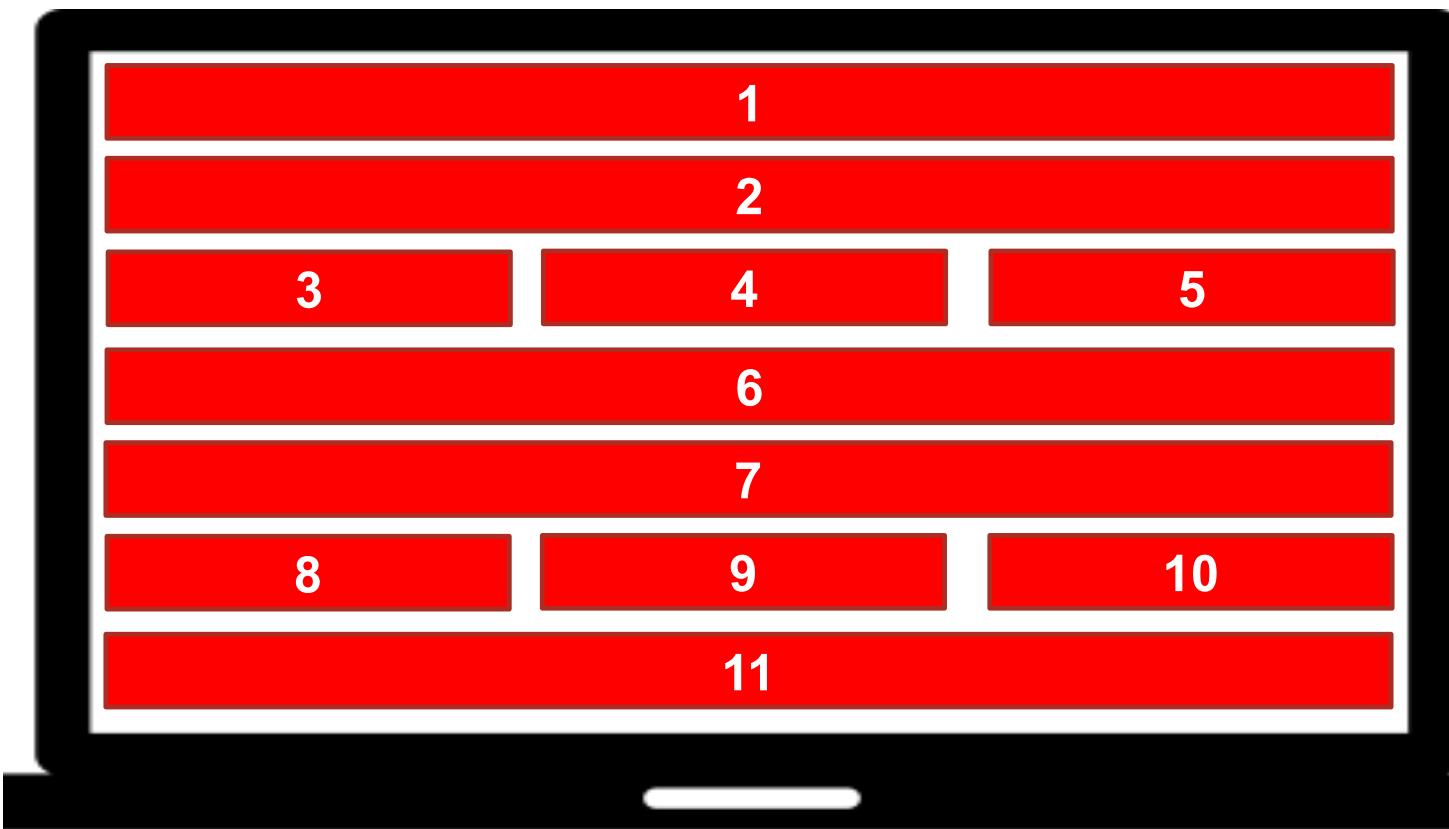


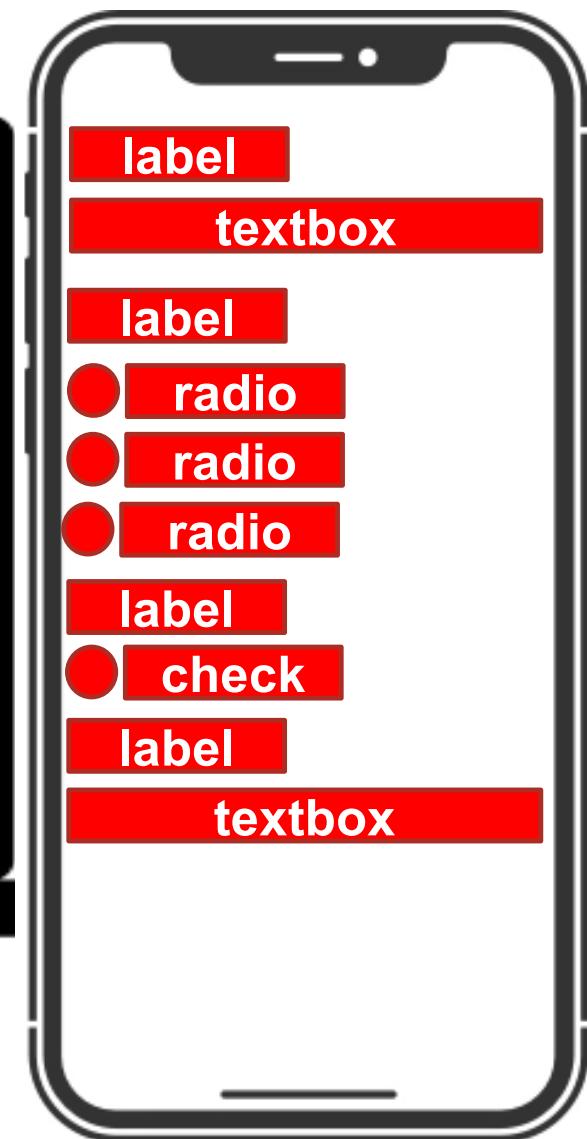
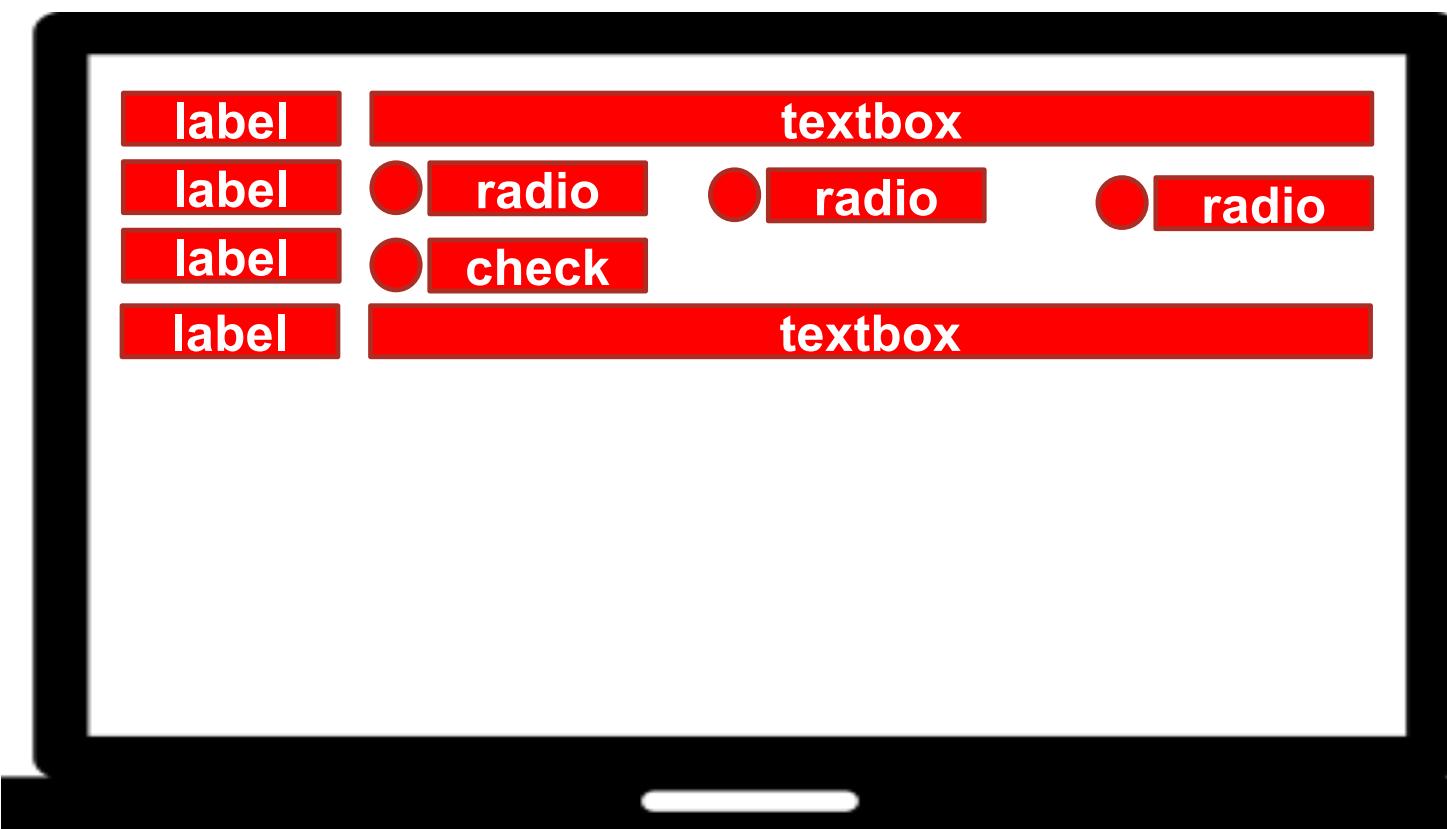
1  
2 3  
4 5  
6  
7 8  
9 10  
11 12



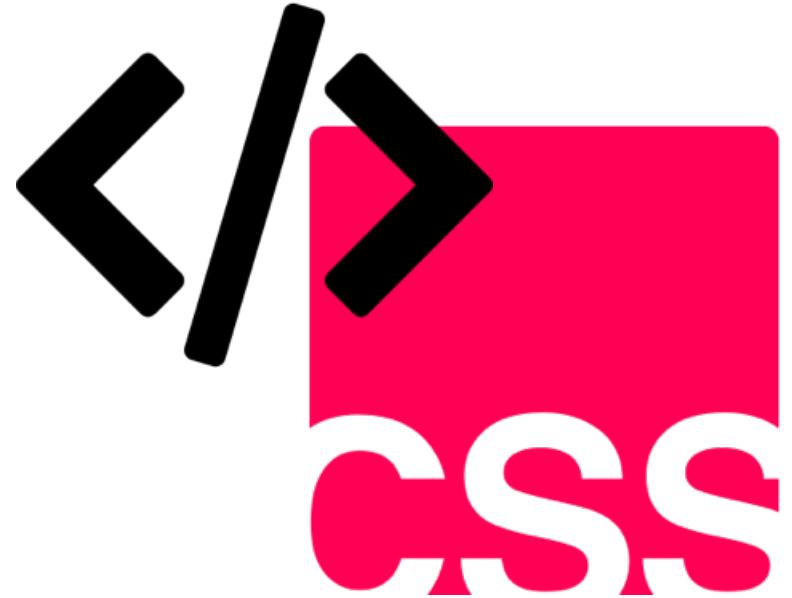








# Progressive Web Apps and Responsive Design



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

## 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

# Programming for mobile devices is difficult



iOS

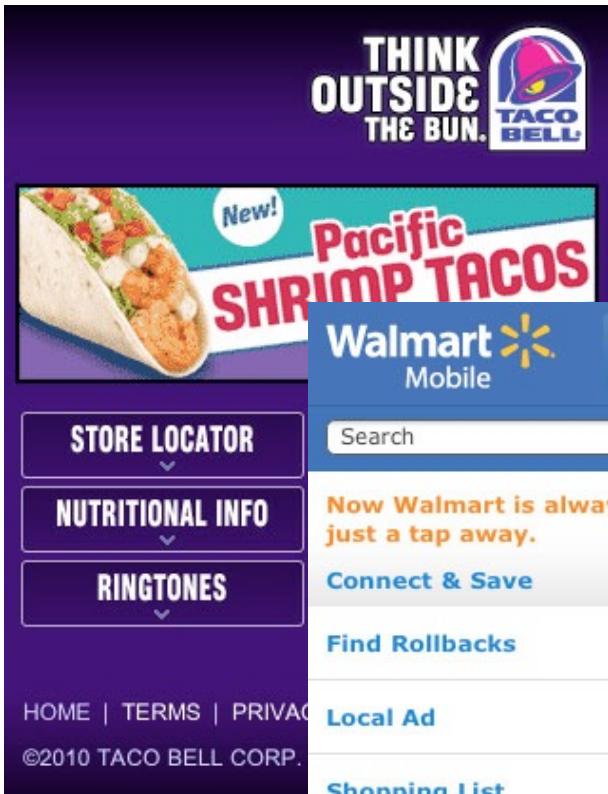
- Objective-C
- Cocoa Touch
- XCode IDE

Android

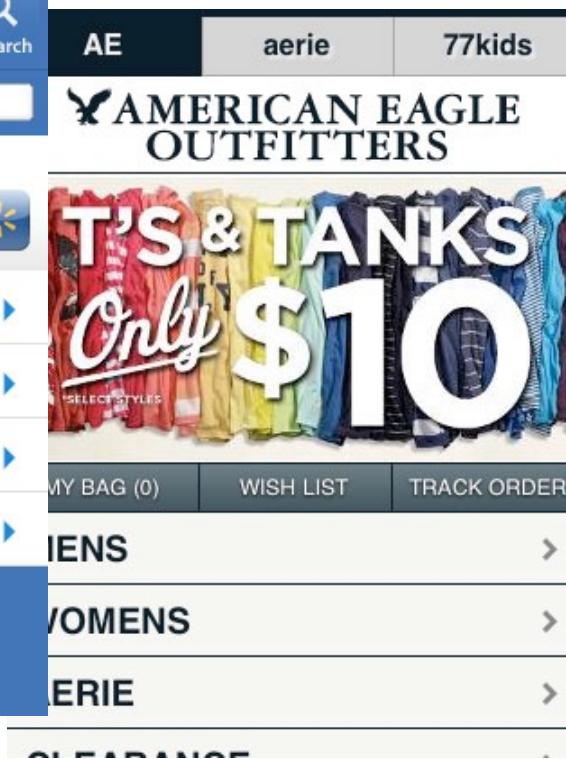
- Java
- ADT plugin
- Eclipse IDE



**And you have to write the app twice! There's no sharing between iOS and Android.**

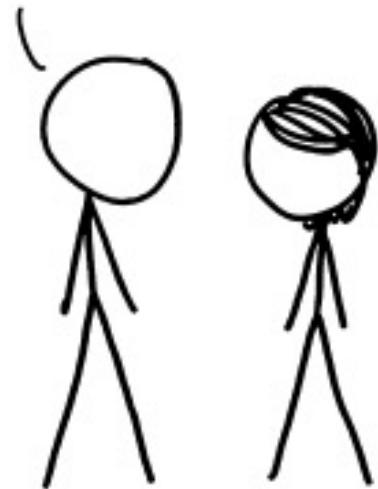


But if we create a web application,  
users can simply ... browse!



INSTALLING THINGS HAS  
GOTTEN SO FAST AND PAINLESS.

WHY NOT SKIP IT ENTIRELY,  
AND MAKE A PHONE THAT HAS  
EVERY APP "INSTALLED" ALREADY  
AND JUST DOWNLOADS AND RUNS  
THEM ON THE FLY?



I FELT PRETTY CLEVER UNTIL I  
REALIZED I'D INVENTED WEBPAGES.

Ummm ...  
This is really  
nothing new

So what are the  
implications?

## Pros

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



**Write once. Run anywhere!**

## 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)

# Progressive web apps

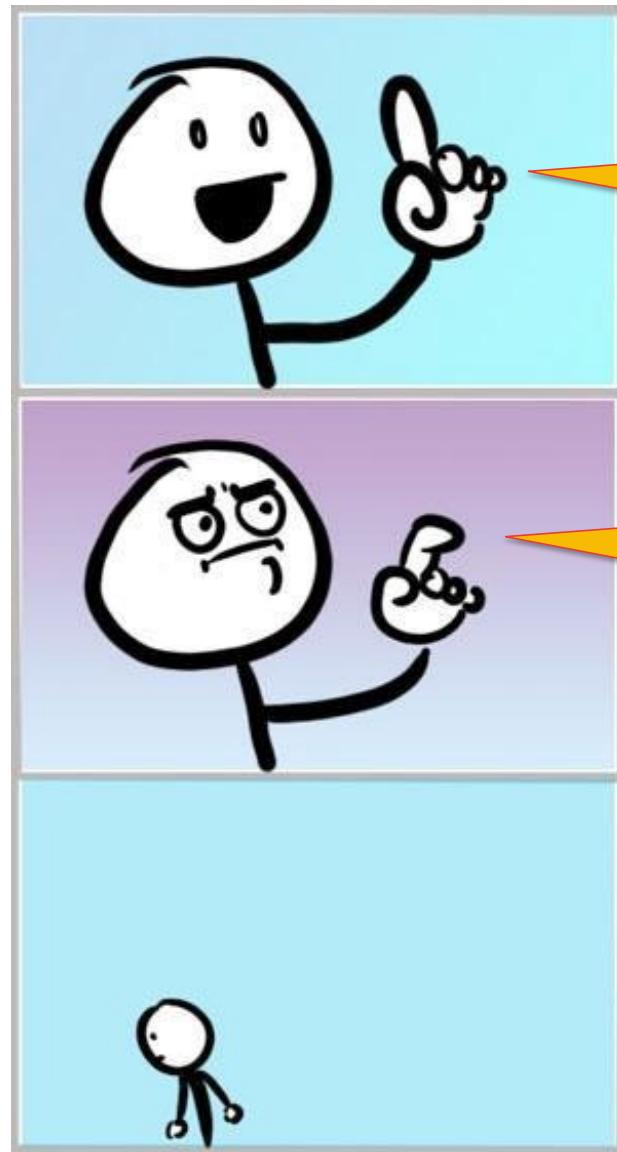
A 10,000 foot view

## What makes a PWA? Everyone agrees that a PWA ...

- Must be served over HTTPS
- Runs offline
- Can have an icon on the home screen

## But some people add these requirements

- 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



The major  
weakness is having  
to be constantly  
connected

**Agile GADGETS**

**Edit your inspection**

**Bill or Credit - Everything**

**Carol Jones CPA (small)**

**Carol Jones**

**QuickBooks Online Essential**

**Home Company Enter Bill**

**Vendor List Purchase Order Enter Bill**

**Enter Bill**

**Recent**

**Pay**

**Full payment**

**Date**

**10/1**

**Address**

**123 Example Lane**

**City**

**Anytown**

**Inspection fee**

**\$338.80**

**save**

**Date:**

**12/21**

**cancel**

**Customer**

**First Name:**

**Highrise**

**Welcome**

**Latest activity**

**Contacts ~60 people**

**Tasks 12 overdue**

**Cases 15 open**

**Deals 2 pending**

**Search notes**

**RECENTLY VIEWED**

**Jordan Chang**

**Madeleine Smith**

**Janice Brown**

**Stephanie Lake**

**Leia Scofield**

**Bank of America**

**James Fallows**

**Marketing Director Search**

**Logo**

**\$7,750 deal w/ Stephanie Lake**

**Marketing Strategy**

**350,000 deal w/ Jordan Chang**

**Federico**

**2**

**Campaigns**

**Lists**

**Reports**

**Autoresponders**

**Search**

**Lists**

**Generitech List 1**

**Stats** **Manage Subscribers** **Signup Forms** **Import** **Settings**

**Segment** **Actions** **View**

**CSV** **Toggle Columns**

|                          | Email Address          | First Name | Last Name  | I'm a ...  | Last Updated  | Date Added        |
|--------------------------|------------------------|------------|------------|------------|---------------|-------------------|
| <input type="checkbox"/> | aarron@generitech.biz  | Aarron     | Waters     | Designer   | Two weeks ago | 3/20/2013 10:06AM |
| <input type="checkbox"/> | jason@generitech.biz   | Jason      | Beards     | Developer  | Two weeks ago | 3/20/2013 10:06AM |
| <input type="checkbox"/> | freddie@generitech.biz | Freddie    | von Chimp  | Boss       | Two weeks ago | 3/20/2013 10:06AM |
| <input type="checkbox"/> | alvaro@generitech.biz  | Alvaro     | Cohen      | Developer  | Two weeks ago | 3/20/2013 10:06AM |
| <input type="checkbox"/> | tyrick@generitech.biz  | Tyrick     | Hobbes     | Designer   | Two weeks ago | 3/20/2013 10:06AM |
| <input type="checkbox"/> | fed@generitech.biz     | Fed        | Scheuneman | Developer  | Two weeks ago | 3/20/2013 10:06AM |
| <input type="checkbox"/> | mardav@generitech.biz  | Mardav     | Brandes    | Developer  | Two weeks ago | 3/20/2013 10:06AM |
| <input type="checkbox"/> | caleb@generitech.biz   | Caleb      | Andrew     | Designer   | Two weeks ago | 3/20/2013 10:06AM |
| <input type="checkbox"/> | gregg@generitech.biz   | Gregg      | Chernov    | Researcher | Two weeks ago | 3/20/2013 10:06AM |

my strong business and marketing foundation would benefit your department, customers and bottom line.

## Our online/offline strategy

- Let's create an experience that convinces the user that they are online when they are 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

## 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

# To go offline, you have to have a manifest

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



**It tells the device how  
to treat this like a PWA**

```
{  
  "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

# Responsive design

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



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

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**

# Responsive & Adaptive both solve this problem

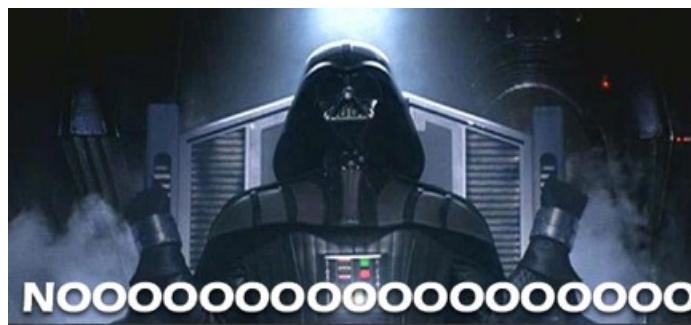
## Responsive Design

- 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



**Adaptive is static. Responsive is liquid.**

**Responsive**



**Adaptive**



We have to separate  
data from  
presentation



All your layout in  
CSS. None in  
HTML

eg. No page layout with  
tables!

You'll create  
new sections in  
your stylesheet,  
one for each  
screen size

### Foo.css

```
@media max-width: 376px {  
    /* styles for small screen  
here */}  
@media max-width: 768px {  
    /* styles for a larger  
screen here */}  
@media max-width: 992px {  
    /* styles for even larger  
screen here */}
```

# 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

# Width is the other key



## 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

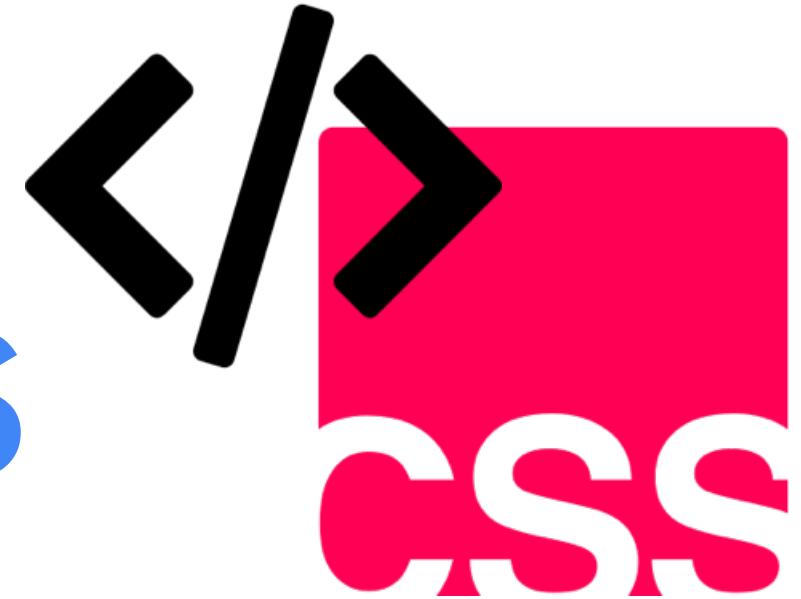
# Create bands of layouts

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

## 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

# Modern CSS Formatting



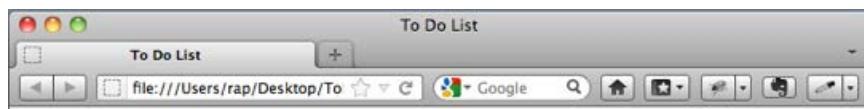
How to make the web look great using  
modern techniques

## 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

# With CSS alone ...

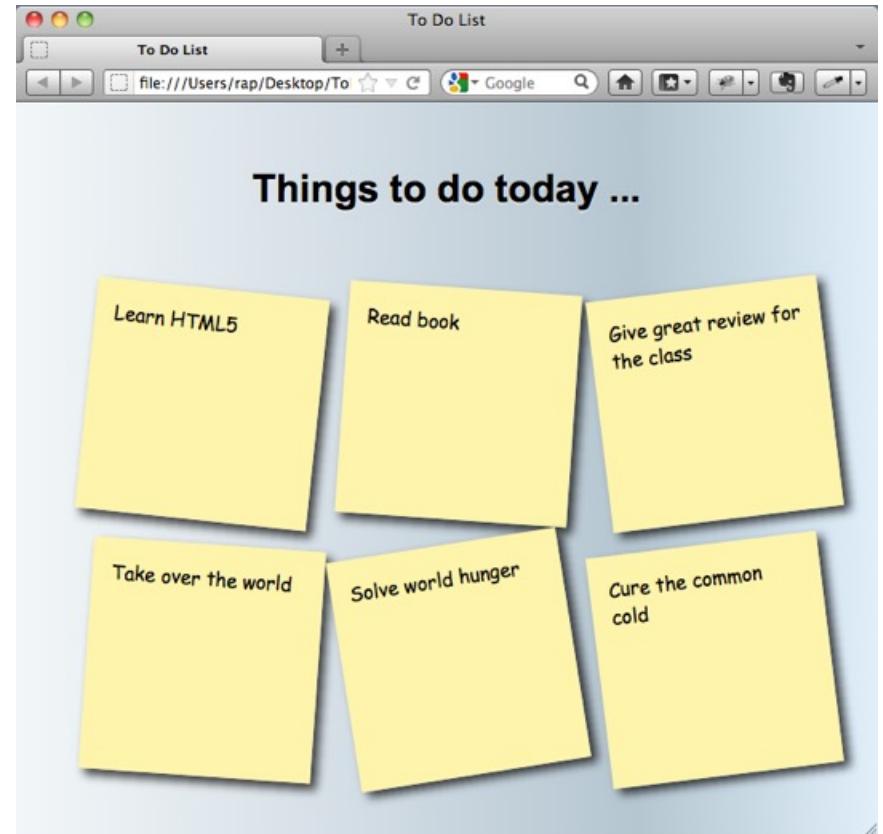
We can take this:



Things to do today ...

- Learn HTML5
- Read book
- Give great review for the class
- Take over the world
- Solve world hunger
- Cure the common cold

And make it look like this:



# Styling Text

## What properties can be set?

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

# 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

# font-style

- font-style:  
normal|italic|oblique

font-style: italic;



# 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

# 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;

# 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;  
}
```

## 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)

## Here's an example

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

# css Tips and Tricks

Who doesn't like tips?

# Backgrounds

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

# When to use <img> vs background?

```

```

```
#someDiv {  
  background: url("foo.jpg");  
}
```

Makes room for 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 a11y

Invisible to a11y

When you want the img to drive the  
layout of the page or when it would be  
important to a11y

When you want the image to be  
decoration for a section that should size  
itself and is unimportant to a11y.

# How do I resize the background image?

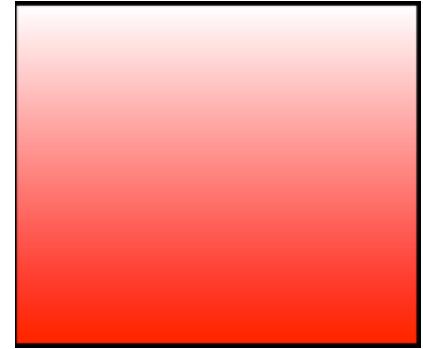


```
#someDiv: {  
background-repeat: no-repeat;  
background-size: cover;  
/* or contain */  
/* or "100px 125px" */  
background-position: 10px 20px;  
/* horizontal vertical;  
  
Can use...  
- "top"  
- "bottom"  
- "center"  
- "left"  
- "right"  
*/  
}
```

# Gradients

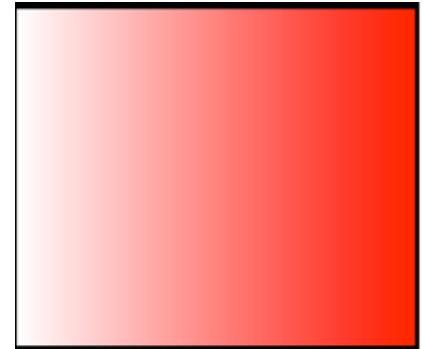
## Top to bottom

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



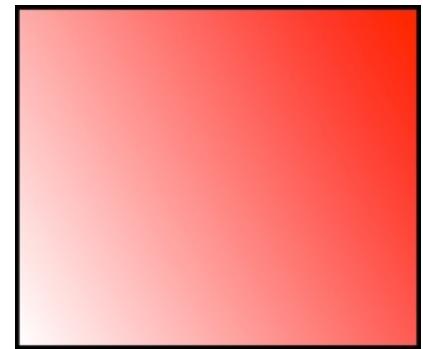
## Left to right

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



## Bottom right to top left

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



# Data uris may speed up your images

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

## Pros

Fewer HTTP requests!

## Cons

Size is increased by 1/3

Browser has to process the image

So what's best practice?



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

## 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 */  
}
```

## CSS Filter property



# box-shadow

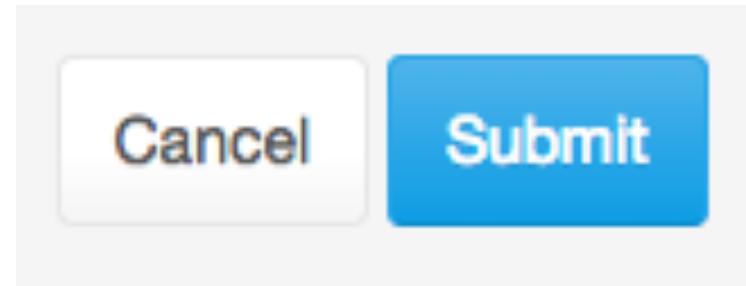
**box-shadow: x y blur color;**



## Rounded corners

5 border-radius: pixels;

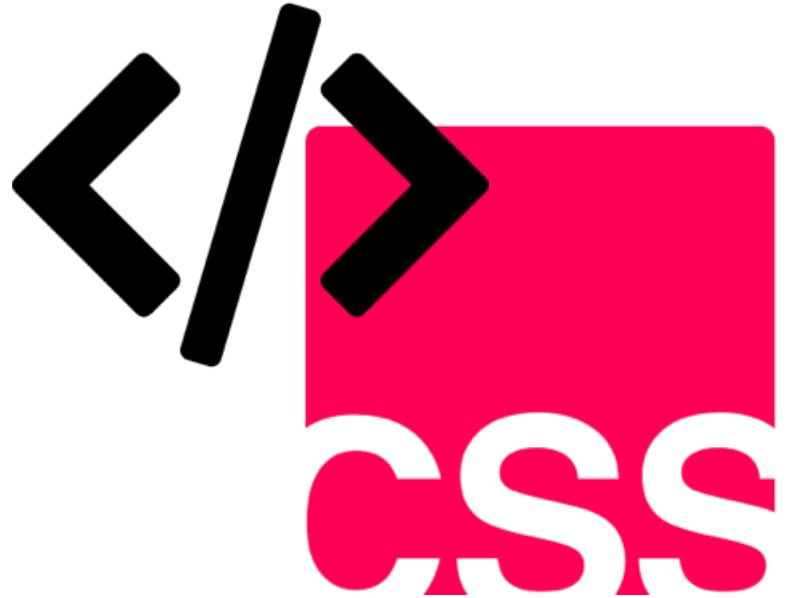
```
.dialog {  
    border-radius: 5px;  
}
```



## 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

# Advanced CSS Selectors



How to point to almost anything

## 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

Remember the basic  
style syntax ...

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



Isn't that  
good enough?

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
- ...

# So why should we learn this stuff?

For ninja-like precision with

- CSS and styles
- `element.querySelector();`
- `element.querySelectorAll();`
- Angular
- jQuery
- Other tools/Frameworks



**⚠ CAUTION**



DROWSINESS ALERT!  
COMPLEX AND DRY  
TECHNICAL MATERIAL  
AHEAD!  
PROCEED WITH CAFFEINE

**Remember your basic  
selectors?**

## Element selectors point to all elements of a certain type

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



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

.className {color: red;}



The period (.) makes it a **class selector**

- 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



# Combining selectors

# 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;  
}
```

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

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

\*or\*

All <p>s  
with a  
class of  
heavy



The first  
thing with  
an id of  
"firstName"

# Grouping selectors can add to 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;}
```

# Relationship selectors

Selecting by DOM position

A photograph of a large family standing in a line on a beach, facing the ocean at sunset. The sky is filled with warm orange and yellow hues. The family members are silhouetted against the bright horizon.

The DOM also points to  
relationships between elements

Descendants Parent Child Siblings Ancestors

# 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;  
}
```

# Child selectors

parent > child

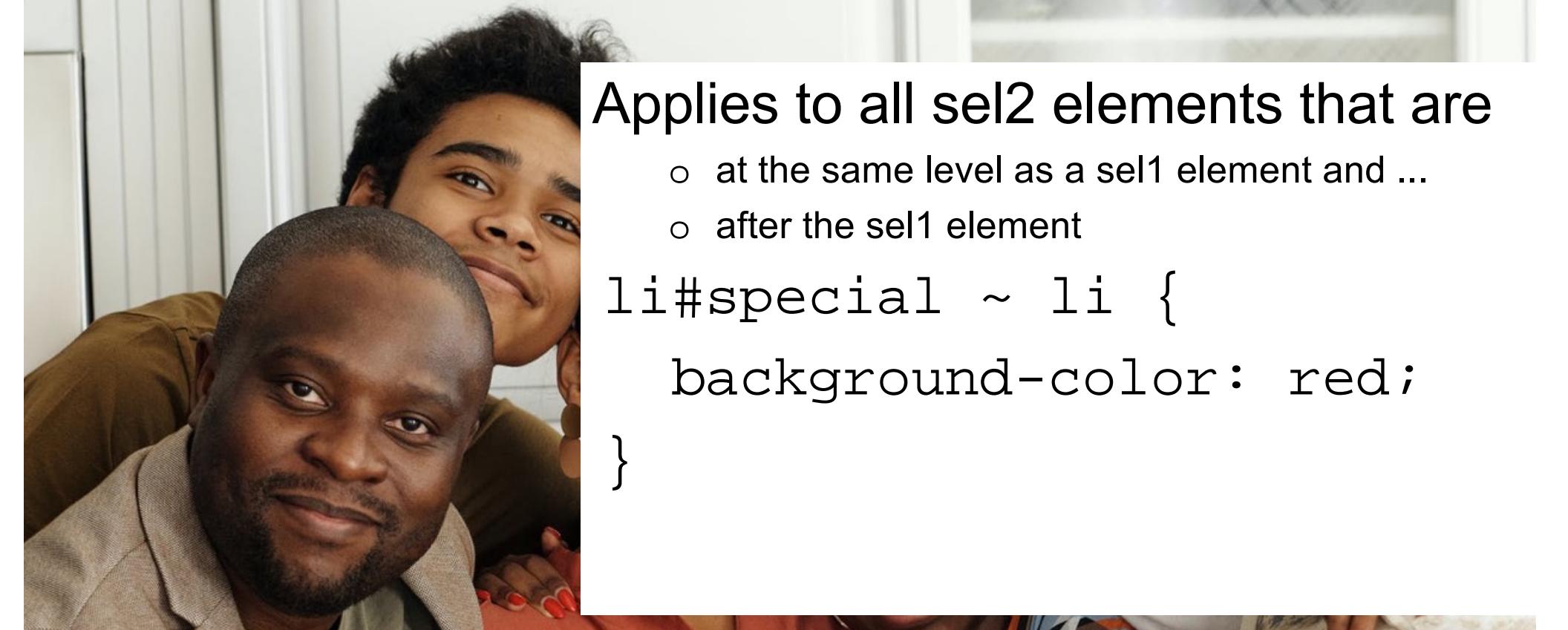
- Selects only **direct** children

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



# 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;  
}
```

## Adjacent sibling selectors

sel1 + sel2

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

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

# Attribute selectors

# 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



**Square brackets ("[" & "]") means it's an attribute selector**

`sel[attr]`

The attribute exists at all. It is *present*

|                       |   |
|-----------------------|---|
| <code>img[alt]</code> | All img tags that have an alt attribute |
| <code>a[href]</code>  | All links that have an href             |
| <code>[id]</code>     | Everything on the page that has an id   |

`sel[attr="value"]`

The attribute is exactly "value". It is set to "value"

|                                     |                                |
|-------------------------------------|--------------------------------|
| <code>img[src="help.gif"]</code>    | All images displaying help.gif |
| <code>input[type="text"]</code>     | All textboxes                  |
| <code>input[type="checkbox"]</code> | All checkboxes                 |

`sel[attr^="value"]`

The attribute starts with "value". It can have other things after it

|                                     |   |
|-------------------------------------|---|
| <code>img[alt^="picture of"]</code> | All img tags that have an alt attribute that begins with "picture of" |
| <code>a[href^="http:"]</code>       | All links whose destination is not using TLS.                         |
|                                     |   |

```
sel[attr$="value"]
```

The attribute ends with "value". It can other things before it

|                        |   |
|------------------------|---|
| a[href\$=".gov"]       | All links to government sites                                       |
| a[href\$="google.com"] | All links to Google's main page regardless of protocol or subdomain |
|                        |   |

`sel[attr*="value"]`

The attribute contains "value" somewhere -- anywhere -- in it

|                                    |  |
|------------------------------------|--|
| <code>img[alt*="tiny"]</code>      | All img tags that have an alt attribute that with "tiny" in it |
| <code>a[href*="google.com"]</code> | All links to Google, no matter how deep the link               |
|                                    |  |

... and there are even more attribute search options but ...



# Pseudo-classes

# Pseudo-classes are like CSS 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

## Pseudo-classes for links

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

# User-action pseudo-classes

- **:hover**
  - We're hovering over it
- **:active**
  - We're clicking on it right now
- **:focus**
  - The thing the user clicks into or tabs into

## 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`

# 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
- ... but wait until you see the super-powerful ...

**:nth-child(something)**

# :nth-child(something)

The *something* can be ...

| X (a number)  |                                |
|---------------|--------------------------------|
| :nth-child(1) | The first child only           |
| :nth-child(5) | The 5 <sup>th</sup> child only |

| Xn (a number followed by "n") |  |
|-------------------------------|--|
| :nth-child(2n)                | 2 <sup>nd</sup> , 4 <sup>th</sup> , 6 <sup>th</sup> , etc.   |
| :nth-child(5n)                | 5 <sup>th</sup> , 10 <sup>th</sup> , 15 <sup>th</sup> , etc. |
| :nth-child(even)              | same as "2n"   |

| Xn+Y (where Y is a number) |  |
|----------------------------|--|
| :nth-child(2n+1)           | 1 <sup>st</sup> , 3 <sup>rd</sup> , 5 <sup>th</sup> , 7 <sup>th</sup> , etc. |
| :nth-child(5n+1)           | 1 <sup>st</sup> , 6 <sup>th</sup> , 11 <sup>th</sup> , etc.                  |
| :nth-child(odd)            | same as "2n+1"   |

| Xn-Y (where Y is a number) |  |
|----------------------------|--|
| :nth-child(2n-1)           | 1 <sup>st</sup> , 3 <sup>rd</sup> , 5 <sup>th</sup> , 7 <sup>th</sup> , etc. |
| :nth-child(5n-1)           | 4 <sup>th</sup> , 9 <sup>th</sup> , 14 <sup>th</sup> , etc.                  |
| :nth-child(odd)            | same as "2n+1"   |



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

"of-type" can be handy too

Simply replace "-child" with "-of-type"

- **:first-of-type**
  - A child, but just the first one of its type
- **:last-of-type**
  - A child, but just the last one of its type
- **:only-of-type**
  - Anything that has no siblings of the same type
- **:nth-of-type(something)**

# 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

# Pseudo-Elements

## Pseudo-elements have a double colon ("::")

- Point to otherwise inaccessible parts of the page.
- Wraps the thing in an ad-hoc inline element

|                |   |
|----------------|---|
| ::first-line   | Only the first line of a paragraph                              |
| ::first-letter | Just the first letter   |
| ::marker       | Just the bullets/numbers on a <li>                              |
| ::selection    | What the user is selecting, like with the mouse or their finger |

But the coolest pseudo-elements are ...

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

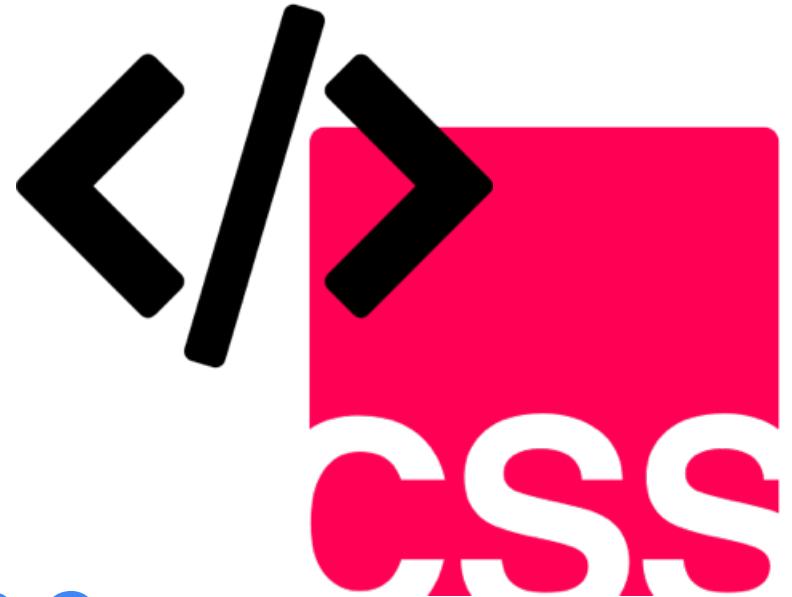
- The only CSS selector that allows you to add *content*
- It wraps the content in an inline element (like a <span>)

```
/* Inserts a logo image in front of every
   <p class="hasLogo"> */
p.hasLogo::before {
  content: url(logo.jpg);
}
/* Adds a " | " after every <li> */
li::after {
  content: " | "
}
```

## 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

# CSS Transforms and Transitions



How to create amazing effects and motion without JavaScript!

## 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

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

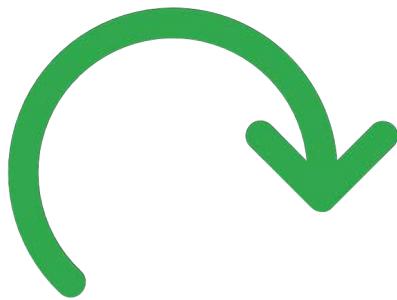
```
<script language="javascript">
var theDiv =
  document.getElementById( "theDiv" );
var y = 5; //StartLocation
var destY = 300; //EndLocation
function moveImage( ) {
  if(y < destY) y = y + 10;
  theDiv.style.top = y +'px';
  if (y + 10 < destY) {
    window.setTimeout(moveImage, 100);
  }
}
</script>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



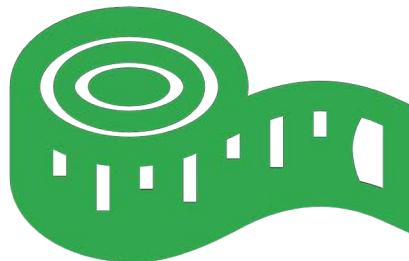
# CSS transforms alter DOM elements, changing their ...



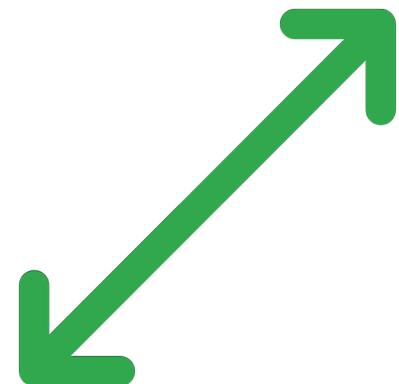
**Location  
(translate)**



**Angle  
(rotate)**



**Size  
(scale)**

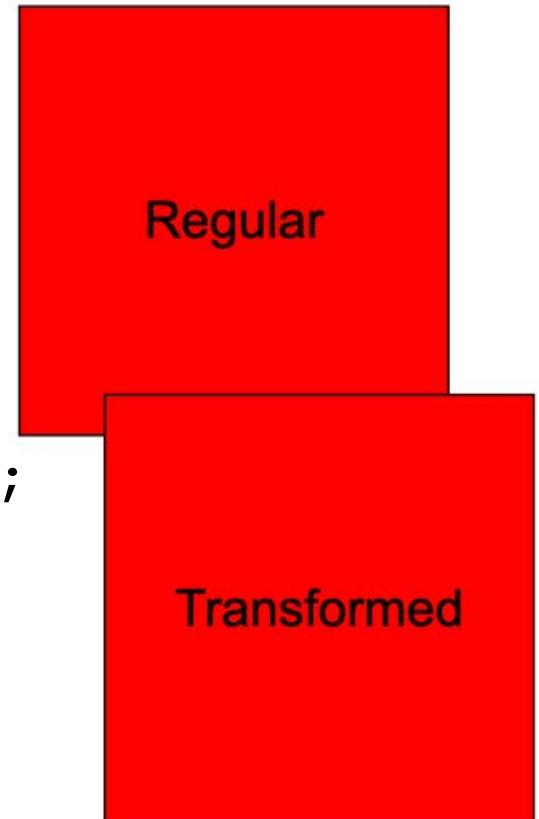


**Distortion  
(skew)**



Location can be  
changed by using  
translate

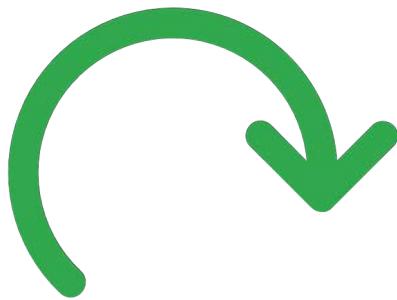
```
div.move:hover {  
    transform: translate(40px, 40px);  
}
```



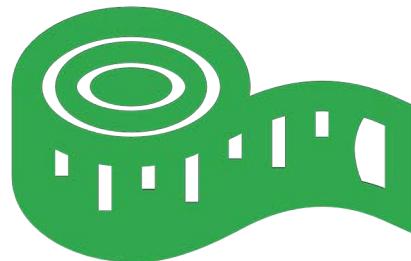
# CSS transforms alter DOM elements, changing their ...



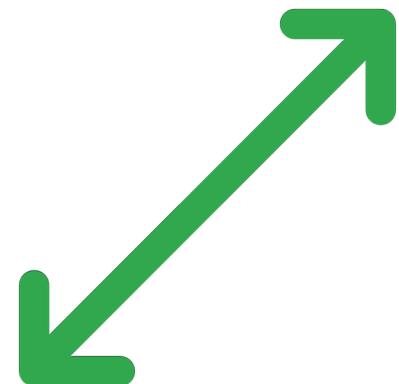
**Location  
(translate)**



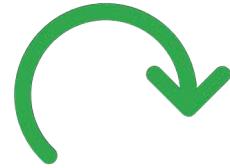
**Angle  
(rotate)**



**Size  
(scale)**



**Distortion  
(skew)**

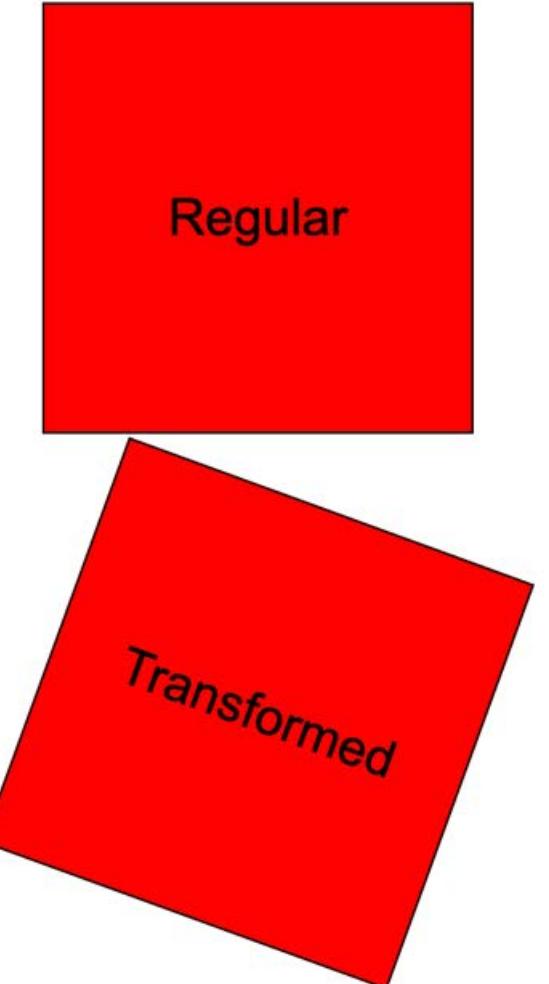


Angle can be  
changed by using  
rotate

```
img.verticalLogo {  
    transform: rotate(20deg);  
}
```



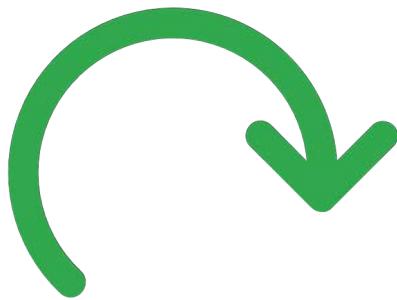
**Angle in deg, rad, grad,  
turn**



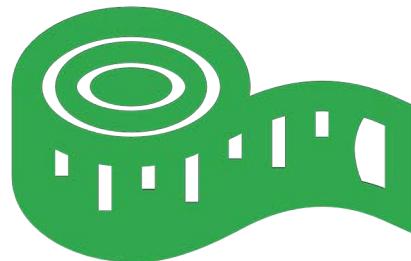
# CSS transforms alter DOM elements, changing their ...



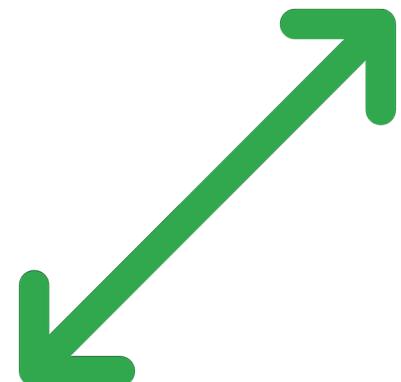
**Location  
(translate)**



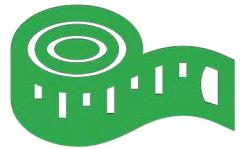
**Angle  
(rotate)**



**Size  
(scale)**

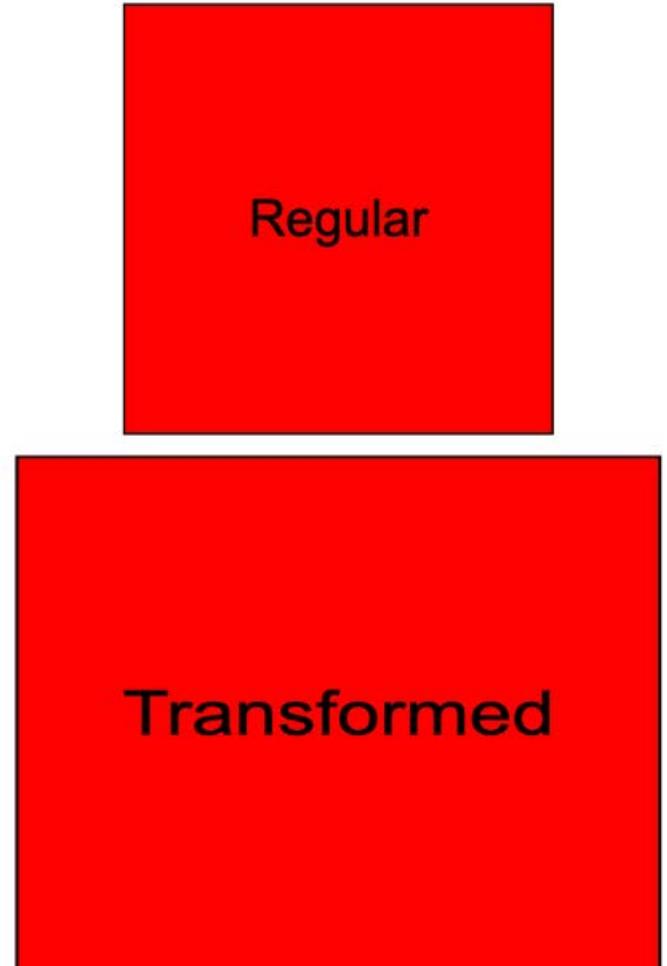


**Distortion  
(skew)**



Resizing can be  
changed by using  
scale

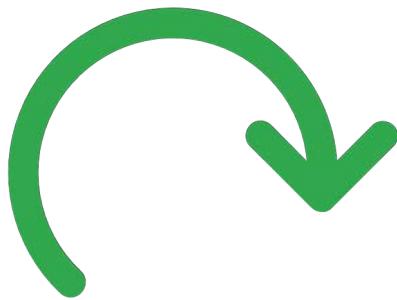
```
#theDiv {  
  transform: scale(1.5, 1.2);  
}
```



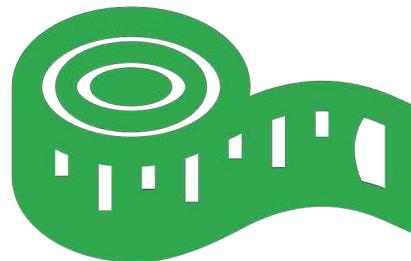
# CSS transforms alter DOM elements, changing their ...



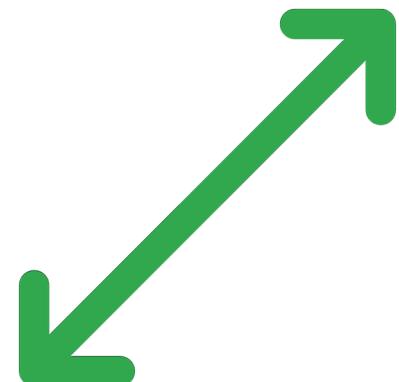
**Location  
(translate)**



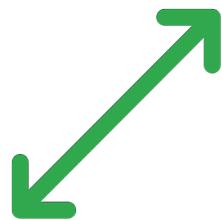
**Angle  
(rotate)**



**Size  
(scale)**

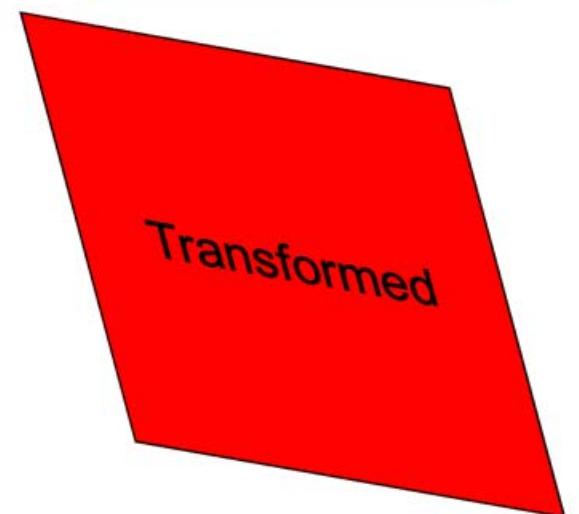
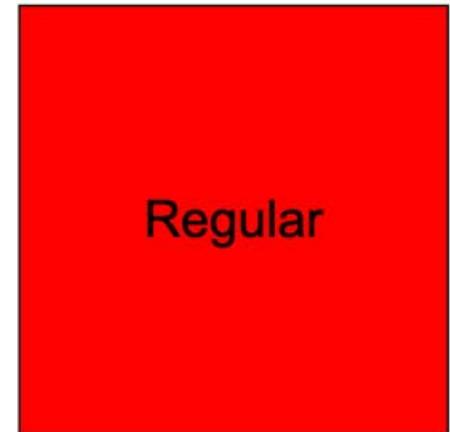


**Distortion  
(skew)**



Distortion can be  
changed by using  
skew

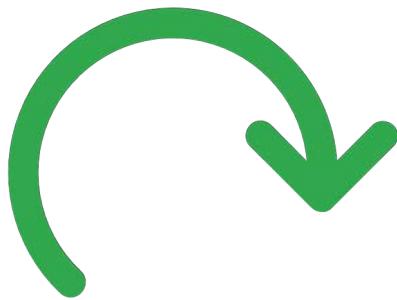
```
#theDiv {  
  transform: skew(15deg, 10deg);  
}
```



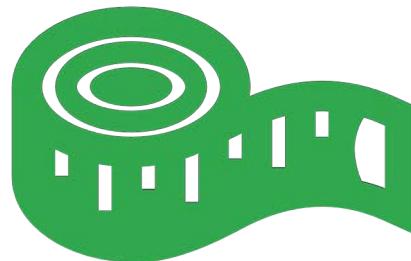
# CSS transforms alter DOM elements, changing their ...



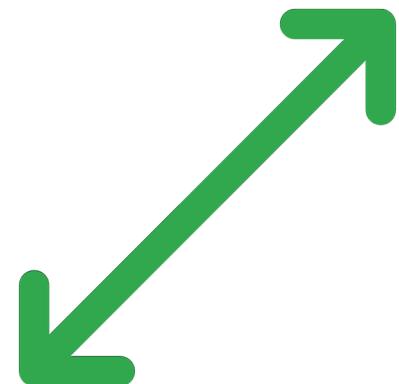
**Location  
(translate)**



**Angle  
(rotate)**



**Size  
(scale)**



**Distortion  
(skew)**

# CSS Transitions

Adding motion to your pages for fun and profit



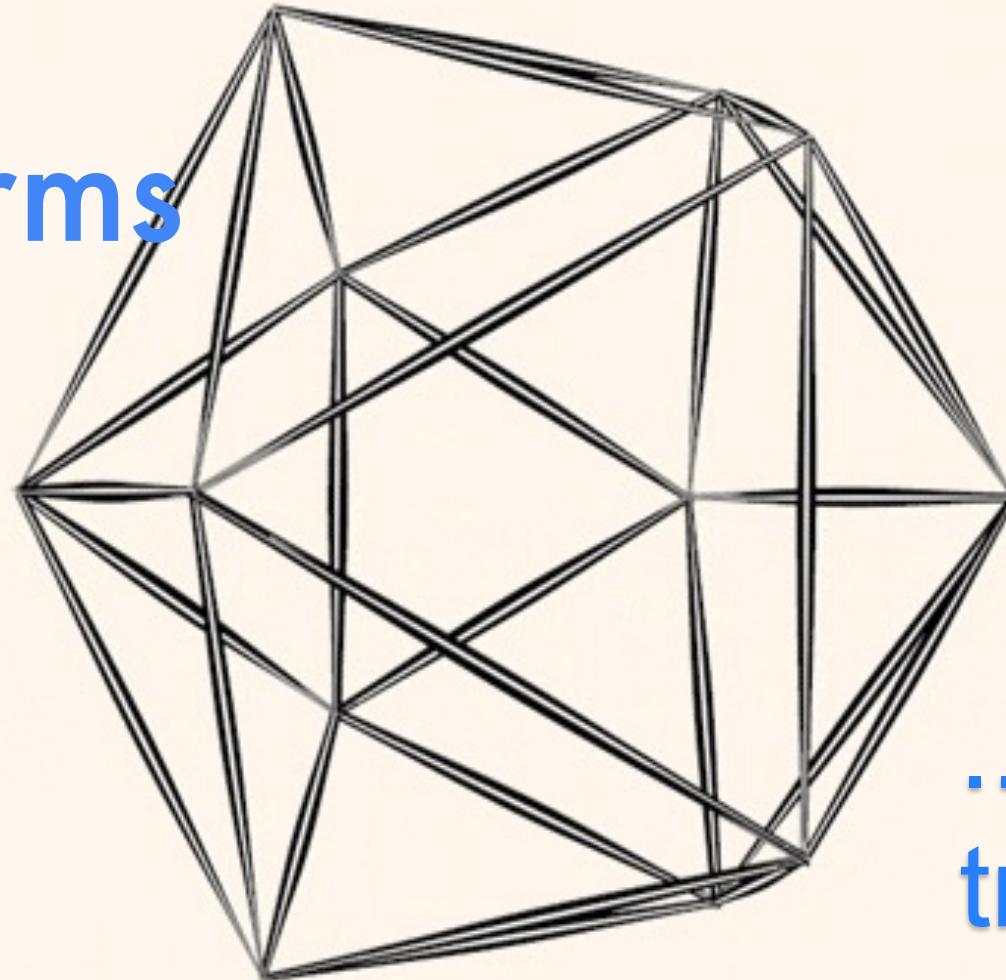
Wait, so  
what are  
"transitions"?

Transforms showed us how to  
change HTML elements

Transitions  
are how to  
change them  
over time

If it  
transforms

...



... it can  
transition

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

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

# Full syntax of transitions

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

```
SomeCssSelector {  
    transition: <property> <duration>  
              <timing-function> <delay>;  
}
```

# The property is the thing you want transitioned

eg. ...

- background-color
  - height
  - width
  - top
  - left
  - transform
- ... or ...
- **all**

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

In seconds or milliseconds

- 5s
- 5000ms

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)



**It's the measure of  
smoothness**

The transition delay tells how long to wait before beginning

- In seconds (s) or milliseconds (ms)

## 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;  
}
```

## We can combine transitions

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

... but this is not okay

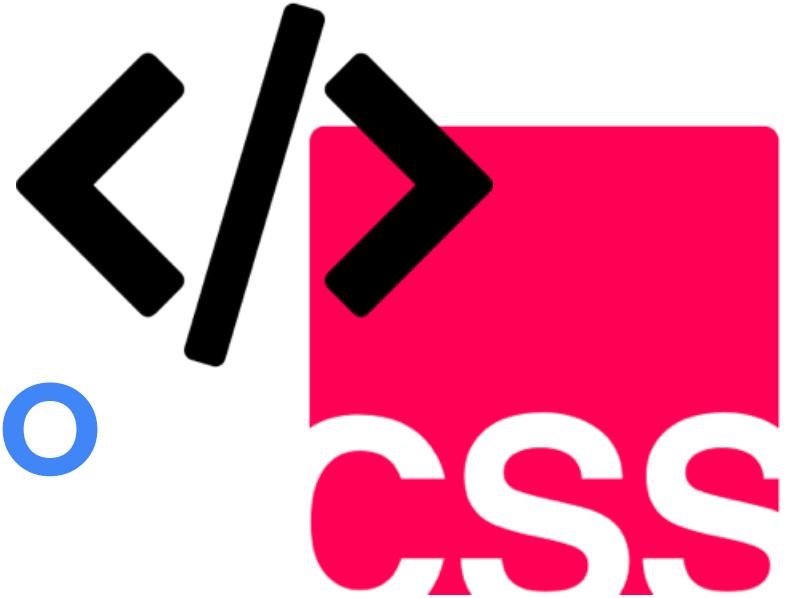
```
#theDiv:hover {  
    transition: top 3s;  
    transition: left 1s;  
}
```

- Why?
- Because the second overwrites the first.

## 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

# Deep Dive into Tables



Presenting data in rows and columns

## tl;dr

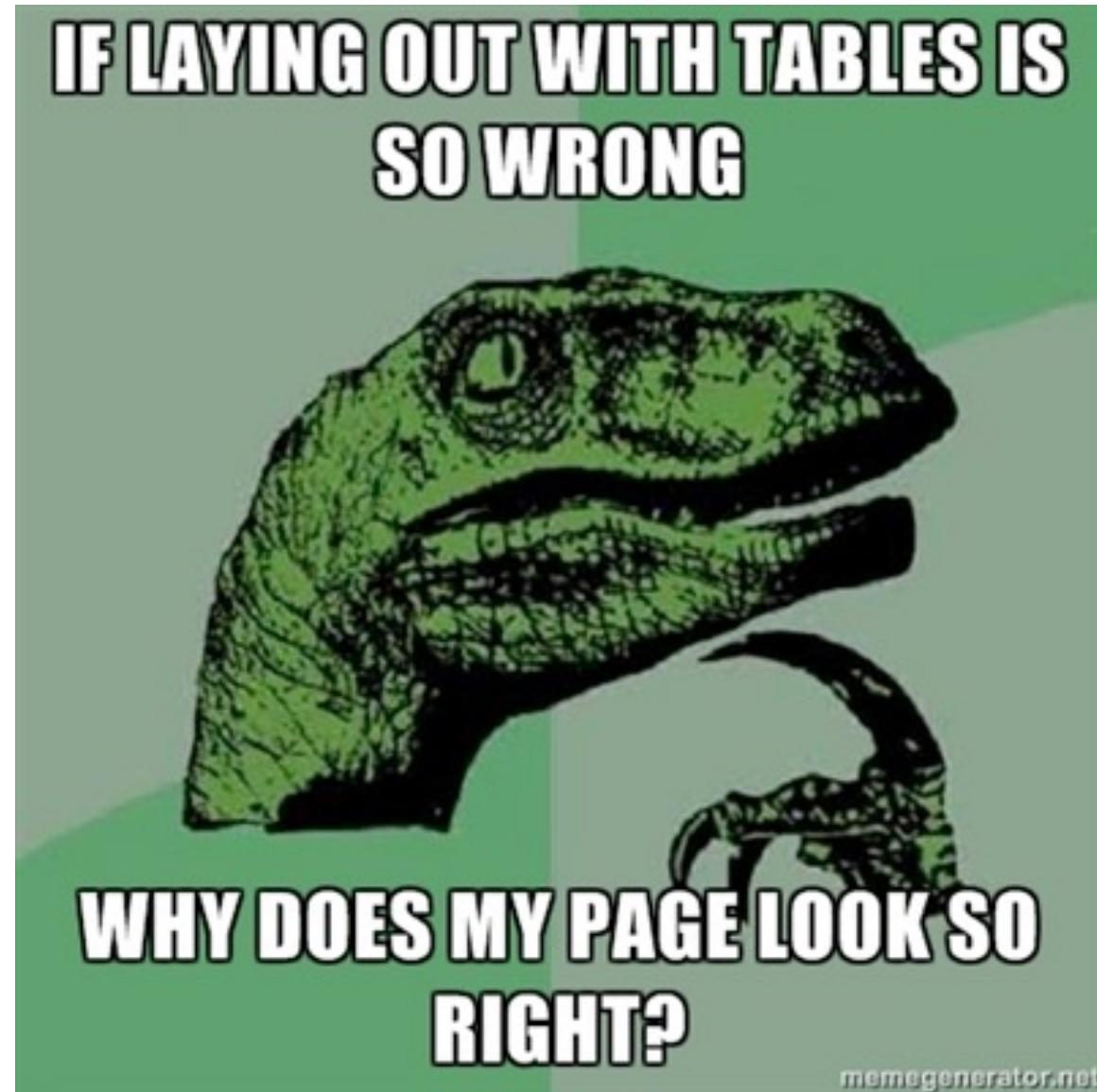
- HTML tables are for data, not for layout
- They can have headers and footers
- Cells can span rows and columns
- We can make tables look fantastic by styling them through CSS

# 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 |

Tables  
are for  
data  
**ONLY**



# Tables are simple to create, but verbose

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



# 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>
```

```
<table>
<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>
</table>
```

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

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

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



```

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;
}

```

## Applying overall styles

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

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 |

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

## Alternating rows

**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 |

# Hovering over a row: tr:hover

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

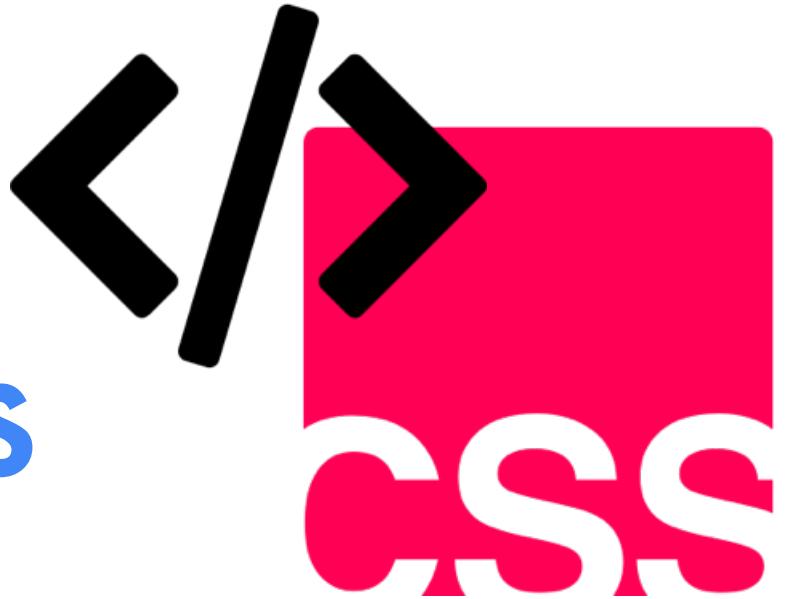
**First Quarter Sales**

|                | 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 |

## 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

# Best Practices with Forms

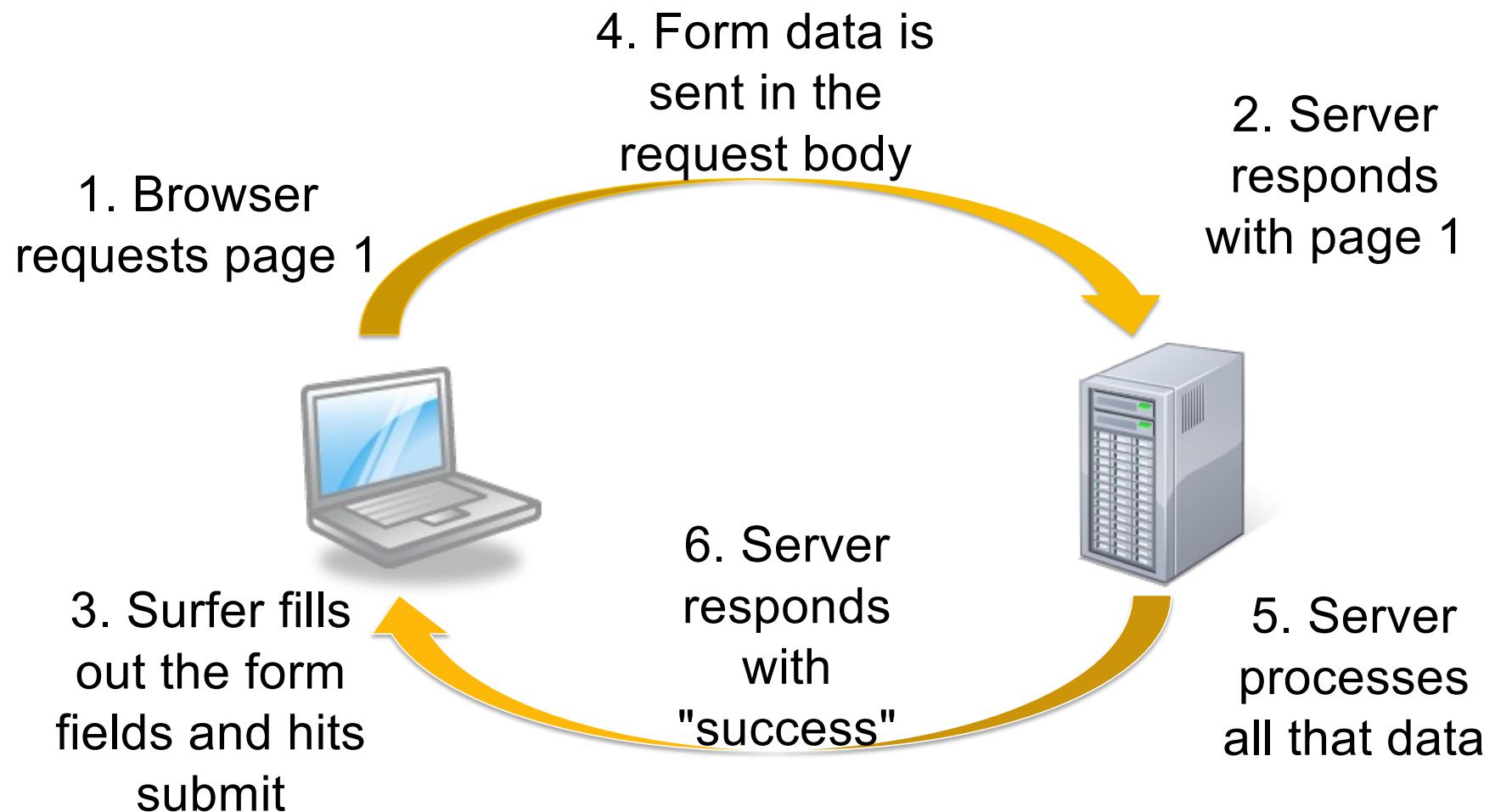


... because a web session is a dialogue,  
not a monologue

## 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

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



# HTTP Methods

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

## The data may look like this...

```
address=2636+Harvard+Road&city=Boston&companyName=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"  
}
```

## The <form> tag

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

Forms  
post their  
data in  
the POST  
request's  
payload



# 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

## 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

## <select> can be listboxes or dropdowns

```
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>
```

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

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

# inputs are for everything else

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

- text 5 email
- password 5 url
- checkbox 5 number
- radio 5 range
- file 5 date
- submit 5 datetime
- button 5 month
- hidden 5 week
- 5 time
- 5 search
- 5 color

# 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

# 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>
```

## 5 input type="email"

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

E-mail (email)

The screenshot shows a web form element for an email address. The input field contains the text "abc". Below the input field, a red box highlights an error message: "abc is not a legal e-mail address". To the right of the input field, there is a "datetime" field with a dropdown menu set to "UTC".

abc

abc is not a  
legal e-mail  
address

datetime)  
:

UTC

# The keyboard changes with a type set

`type="text"`



`type="number"`



`type="tel"`



`type="email"`



## 5 input type="number"

- A number

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

## input type="range"

- Also a number

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

Range (range)



## 5 <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  | 11  |
| 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

# Form attributes

# HTML5 attributes

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



Autofocus is the default  
starting point

```
<input name="address"  
      autofocus />
```

## 5 Placeholder puts ghost text in the textbox

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

First Name \*

First Name

Last Name \*

Last Name

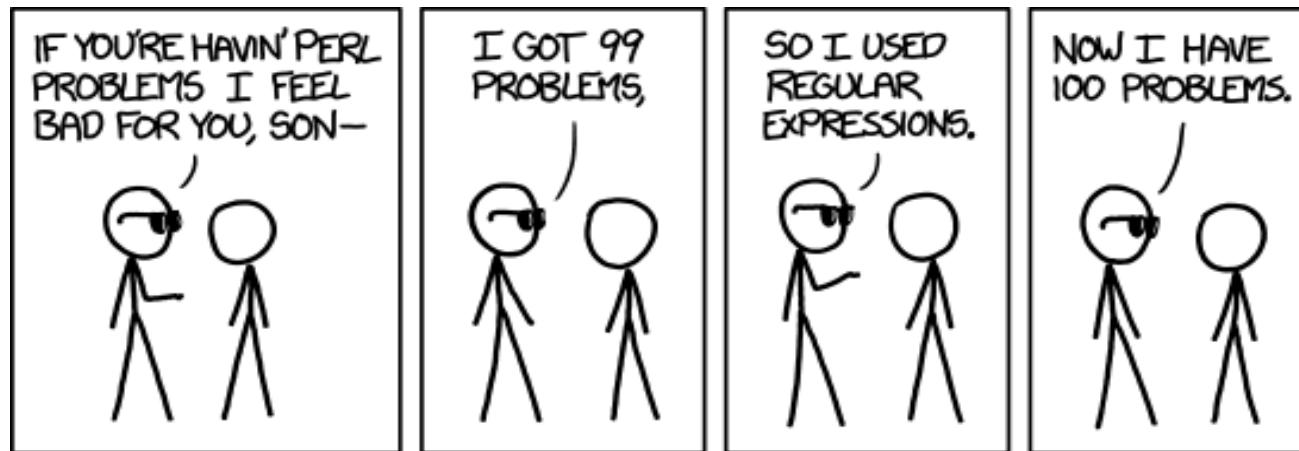
Email address \*

anything@example.com

## 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)' />
```



## **5** Requiring values

- Add "required" to any form element

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

## 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