

# CS193X: Web Programming Fundamentals

Spring 2017

Victoria Kirst  
(vrk@stanford.edu)

# Today's schedule

## Today

- position
- Random helpful CSS
  - calc, variables, background properties
- Mobile layouts
  - em and rem

## Monday

- Intro to JavaScript

"I can think of 12 different ways to implement this!  
Which one is best?"

# Simplicity above all else

Always prefer **simplicity**.

Other tips:

- **Separation of concerns:** HTML should contain content NOT style, CSS should contain style NOT content
- **Descriptive HTML tags:** Make your HTML more readable by using e.g. `<header>` instead of `<div>` when appropriate
- **Reduce redundancy:** Try grouping styles, using descendant selectors to reduce redundancy (see past slides for details)

# Last time

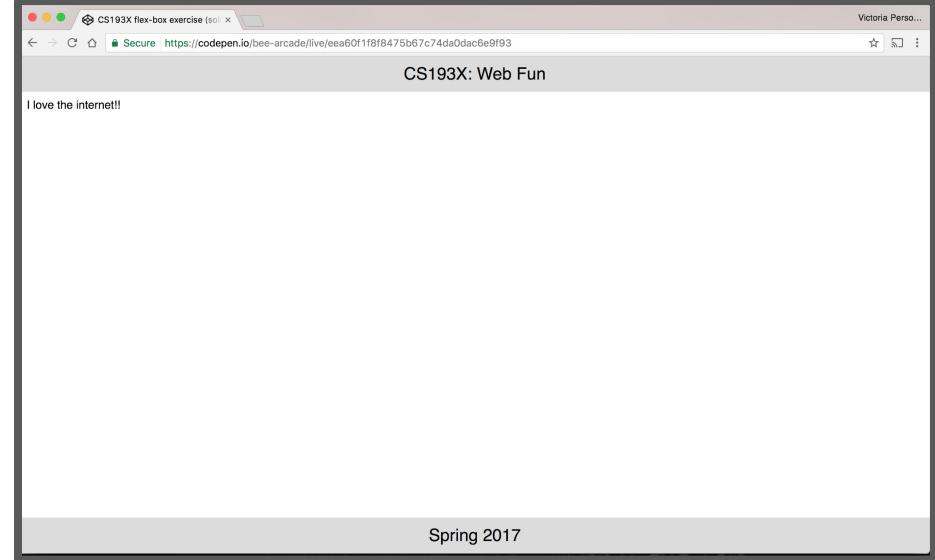
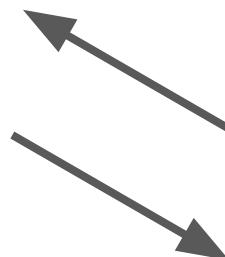
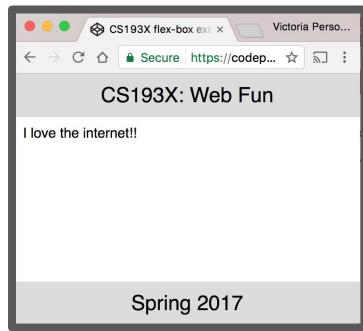
More flexbox stuff:

- **flex-shrink**
  - Default value is 1 (**on** by default)
  - This CSS property is on the flex *item*, not container
- **flex-grow**
  - Default value is 0 (**off** by default)
  - This CSS property is on the flex *item*, not container

vh / vw / box-sizing

# Flexbox example

How do we make a layout that looks like this? ([Codepen](#))



The header and footer  
stay at the top and  
bottom of the viewport.  
([Live example](#))

# Flexbox example solution

```
article {  
  display: flex;  
  flex-direction: column;  
  height: 100vh;  
}  
  
section {  
  flex-grow: 1;  
  padding: 10px;  
}
```

```
footer,  
header {  
  display: flex;  
  justify-content: center;  
  align-items: center;  
  font-size: 24px;  
  background-color: gainsboro;  
  height: 50px;  
}
```

([Solved CodePen](#))

Another rendering  
mode: position

# Moving things with position

**Positioned layout** lets you define precisely where an element should be in the page ([mdn](#)).

You can use positioned layout doing the following:

1. Define a **position** method:  
Static, fixed, absolute, relative
2. Define **offsets**: top, left, bottom, and right
3. (optional) Define **z-index** for overlapping layers ([mdn](#))

**Let's check it out!**

# Moving things with position

To specify exactly where an element goes, set its **top**, **left**, **bottom**, and/or **right** offset.

The meaning of these offset values depend on the reference point set by **position**:

- **static**: no reference point; static block can't move (*this is the default style for every element*)
- **fixed**: a fixed position within the viewport
- **absolute**: a fixed position within its "containing element"
- **relative**: offset from its normal static position

# position: static

(nothing happens!)

- `static` is the default value for position
- If you use `top / left / bottom / right` without setting a non-static position, nothing will happen

```
<body>
  <h1>Puppy</h1>
  <p>A puppy is a juvenile dog. Some puppies
  <h2>Development</h2>
  <p>At first, puppies spend the large major-
  <div id="box1"></div>
</body>
```

```
#box1 {
  height: 100px;
  width: 100px;
  background-color: red;

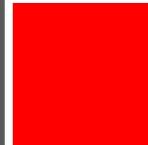
  top: 0;
  left: 0;
}
```

## Puppy

A puppy is a juvenile dog. Some puppies can weigh 1–3 lb up to 15–23 lb (6.8–10.4 kg). All healthy puppies grow quickly as they change as the puppy grows older, as is commonly seen in vernacular English, puppy refers specifically to dogs, while other animals such as seals, giraffes, guinea pigs, or even rats.

## Development

At first, puppies spend the large majority of their time sleeping and playing together in a pile, and become distressed if separated from the group by even a short distance.

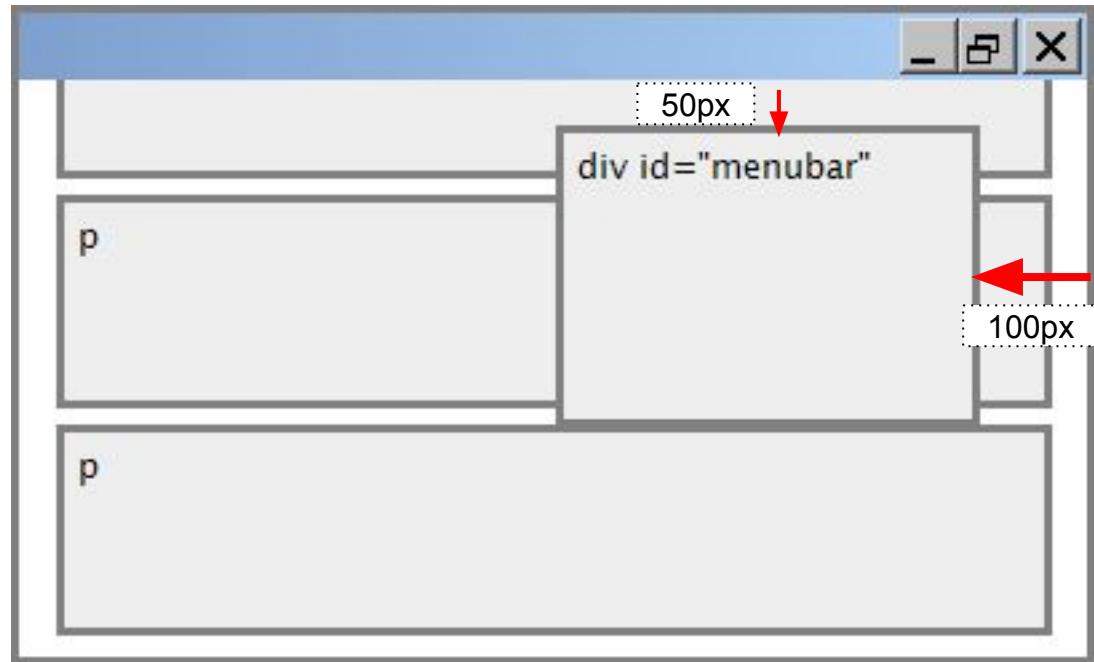


[\(CodePen\)](#)

# position: fixed

```
#menubar {  
    position: fixed;  
    top: 50px;  
    right: 100px;  
}
```

- For **fixed positioning**, the offset is the distance positioned relative to the viewport.
- The element does not move when scrolled.
- Element is **removed from normal document flow**, positioned on its own layer



Often used to implement  
UIs; control bars that  
shouldn't go away

# position: fixed

```
#box1 {  
    height: 50px;  
    background-color:  
        rgba(0, 0, 0, 0.5);  
  
    position: fixed;  
    top: 50%;  
    left: 0;  
    right: 0;  
}
```

([CodePen](#))

vernacular English, puppy refers specifically to dogs, while pup may often be used for other mammals such as seals, giraffes, guinea pigs, or even rats.

## Development

At first, puppies spend the large majority of their time sleeping and the rest feeding. They instinctively pile together into a heap, and become distressed if separated from physical contact with their littermates, by even a short distance.

Puppies are born with a fully functional sense of smell but can't open their eyes. During their first two weeks, a puppy's senses all develop rapidly. During this stage the nose is the primary sense organ used by puppies to find their mother's teats, and to locate their littermates, if they become separated by a short distance. Puppies open their eyes about nine to eleven days following birth. At first, their retinas are poorly developed and their vision is poor. Puppies are not able to see as well as adult dogs. In addition, puppies' ears remain sealed until about thirteen to seventeen days after birth, after which they respond more actively to sounds. Between two and four weeks old, puppies usually begin to growl, bite, wag their tails, and bark.

Puppies develop very quickly during their first three months, particularly after their eyes and ears open and they are no longer completely dependent on their mother. Their coordination and strength improve, they spar with their littermates, and begin to explore the world outside the nest. They play wrestling, chase, dominance, and tug-of-war games.

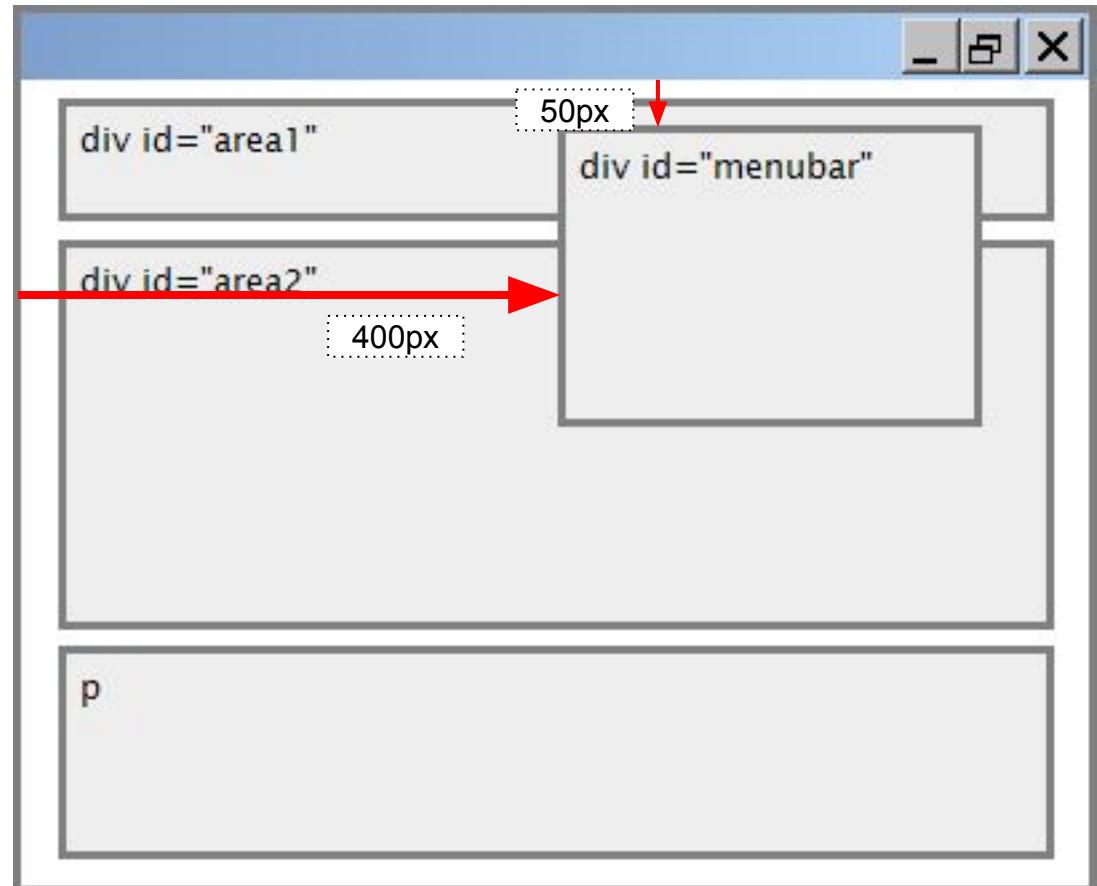
## Development

Puppies are highly social animals and spend most of their waking hours interacting with either their mother or littermates. When puppies are socialized with humans, particularly between the ages of eight and twelve weeks, they

# position: absolute

```
#menubar {  
    position: absolute;  
    left: 400px;  
    top: 50px;  
}
```

- For **absolute positioning**, the offset is the distance from the "containing element", which is the html element by default
- Element is removed from normal document flow, positioned on its own layer



# position: absolute

```
#box1 {  
    height: 100px;  
    width: 100px;  
    background-color: red;  
  
    position: absolute;  
    top: 0;  
    left: 0;  
}  
  
#box2 {  
    height: 100px;  
    width: 100px;  
    background-color: blue;  
  
    position: absolute;  
    top: 50px;  
    left: 50px;  
}
```



can weigh up to 23 lb (6.8–10.4 kg). All healthy puppies grow quickly after birth. A puppy's coat color may change as the puppy grows older, as is commonly seen in breeds such as the Yorkshire Terrier. In vernacular English, puppy refers specifically to dogs, while pup may often be used for other mammals such as seals, giraffes, guinea pigs, or even rats.

## Development

At first, puppies spend the large majority of their time sleeping and the rest feeding. They instinctively pile together into a heap, and become distressed if separated from physical contact with their littermates, by even a short distance.

# position: relative

For `position: relative;` the element is placed where it would normally be placed in the layout of the page, but shifted by the `top` / `left` / `bottom` / `right` values.

```
#box2 {  
  height: 100px;  
  width: 100px;  
  background-color: blue;  
  
  position: relative;  
  top: 50px;  
  left: 50px;  
}
```

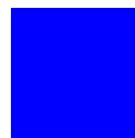
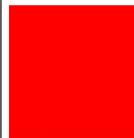
([CodePen](#))

## Puppy

A puppy is a juvenile dog. Some puppies can weigh 1–3 lb (0.45–1.36 kg), while larger ones can weigh up to 15–23 lb (6.8–10.4 kg). All healthy puppies grow quickly after birth. A puppy's coat color may change as the puppy grows older, as is commonly seen in breeds such as the Yorkshire Terrier. In vernacular English, puppy refers specifically to dogs, while pup may often be used for other mammals such as seals, giraffes, guinea pigs, or even rats.

## Development

At first, puppies spend the large majority of their time sleeping and the rest feeding. They instinctively pile together into a heap, and become distressed if separated from physical contact with their littermates, by even a short distance.



# Relative absolute positioning

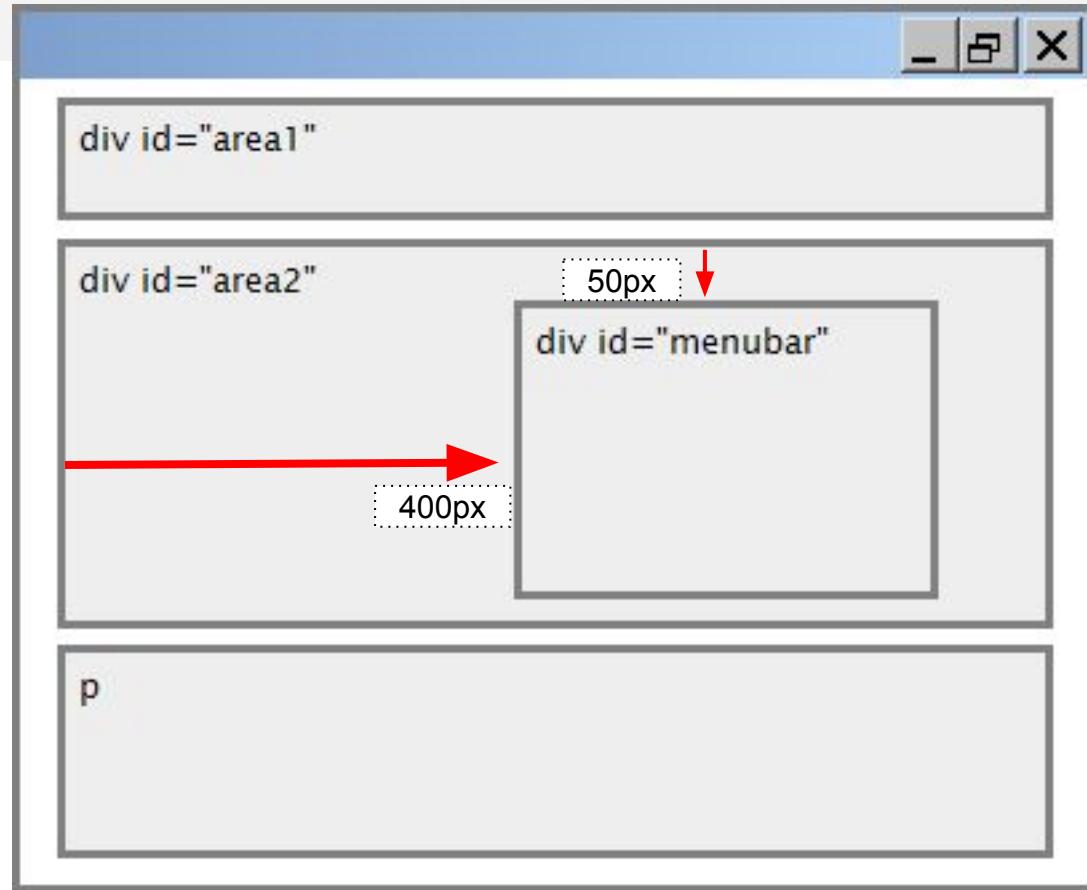
Let's revisit the definition of absolute positioning:

- **absolute**: a fixed position within its "containing element"
- The containing element is the viewport by default

You can change the containing element by setting "**position: relative;**" on some parent of your absolutely positioned element!

# Relative absolute positioning

```
#area2 {  
    position: relative;  
}  
  
#menubar {  
    position: absolute;  
    left: 400px;  
    top: 50px;  
}
```



Offsets are relative to the first parent element that has **position: relative** which in this case is **#area2**

# Common use case: Overlay

```
<header>
  <div id="overlay"></div>
</header>
```

```
header {
  background-image: url(https://);
  background-size: cover;
  height: 300px;
  position: relative;
}

#overlay {
  background-color:
    rgba(0, 0, 0, 0.3);
  position: absolute;
  top: 0;
  bottom: 0;
  height: 100%;
  width: 100%;
}
```



([CodePen](#))

Let's revisit Squarespace again!  
([link to solution](#))

# Random useful CSS

# calc

You can use the [calc](#) CSS function to define numeric values in terms of expressions:

```
width: calc(50% - 10px);
```

```
width: calc(100% / 6);
```

([MDN details of calc](#))

# CSS variables

**Variables** are a brand-new CSS feature ([caniuse](#)).

```
:root {  
    --primary-color: hotpink;  
}  
  
h1 {  
    background-color: var(--primary-color);  
}
```

([MDN details of CSS variables](#))

# background properties

An easy way to render images stretched and cropped to a given size: set it as a background image for an element.

```
background-image: url(background.png);
```

The screenshot shows a web development interface with two tabs: 'HTML' and 'CSS'. The 'HTML' tab contains the following code:

```
</html>
<body>
  <header></header>
</body>
</html>
```

The 'CSS' tab contains the following code:

```
header {
  background-image: url(https://s3-us-west-2.amazonaws.com/s.cdpn.io/1083533/header.jpg);
  height: 200px;
}
```

Below the code editor is a preview window showing a blue-tinted landscape image, which is the background image applied to the header element.

([CodePen](#))

# background properties

You can then use [additional background properties](#) to further style it:

```
background-size: cover;  
background-size: contain;  
background-repeat: no-repeat;  
background-position: top;  
background-position: center;
```

([CodePen](#): Try resizing the window)

# Web Fonts

You can use [Google Fonts](#) to choose from better fonts:

The screenshot shows the Google Fonts website interface. At the top, there's a navigation bar with links for DIRECTORY, FEATURED, and ABOUT. On the right, there's a search bar and a sidebar with categories like Serif, Sans Serif, Display, Handwriting, and Monospace, all of which are checked. Below the navigation, it says "Viewing 818 of 818 font families". There are five font families displayed in cards:

- Roboto** by Christian Robertson (12 styles). Preview text: "All their equipment and instruments are alive."
- Asar** by Sorkin Type (1 style). Preview text: "A red flair silhouetted the jagged edge of a wing."
- Open Sans** by Steve Matteson (10 styles). Preview text: "I watched the storm, so beautiful yet terrific."
- Lato** by Łukasz Dziedzic (10 styles). Preview text: "Almost before we knew it, we had left the ground."
- Slabo 27px** by John Hudson (1 style). Preview text: "A shining crescent far beneath the flying vessel."

A modal window is open in the center, prompting users to "Try typing directly into the text fields." A "GOT IT" button is visible at the bottom of this modal.

# Web Fonts

You can use [Google Fonts](#) to choose from better fonts:

The instructions are pretty self-explanatory:

Basically, add the given `<link>` tag into the `<head>` section of your page alongside your other CSS files.

The screenshot shows the Google Fonts interface with the following details:

- 1 Family Selected:** Roboto
- Your Selection:** Roboto
- Embed vs Customize:** EMBED (selected)
- Load Time:** Fast
- Embed Font:** Instructions to embed the font into an HTML document by copying the provided `<link>` tag.
- Code Snippet (STANDARD):** `<link href="https://fonts.googleapis.com/css?family=Roboto" rel="stylesheet">`
- Specify in CSS:** Instructions to use the following CSS rule to specify the font family.
- Code Snippet (CSS):** `font-family: 'Roboto', sans-serif;`
- Additional Information:** A link to the [getting started guide](#).

# Aside: Fallback fonts

Notice that the Google Font example shows a comma-separated list of values for `font-family`:

```
font-family: 'Roboto', sans-serif;
```

- Each successive font listed is a fallback, i.e. the font that will be loaded if the previous font could not be loaded
- There are also six [generic font names](#), which allows the browser to choose the font based on intent + fonts available on the OS.
- It's good practice to list a generic font at the end of all your `font-family` declarations.

# Hosted fonts with @font-face

You can also load your own font via @font-face:

- Give it your own font name
- Link to where the font file is found

```
<body>
  <h1>Always and Forever</h1>
</body>
```

```
@font-face {
  font-family: "My Custom Font";
  src: url("https://s3-us-west-2.amazonaws.com/just-another-website/fonts/MyCustomFont.woff");
}

body {
  font-family: "My Custom Font", serif;
}
```

Always and Forever

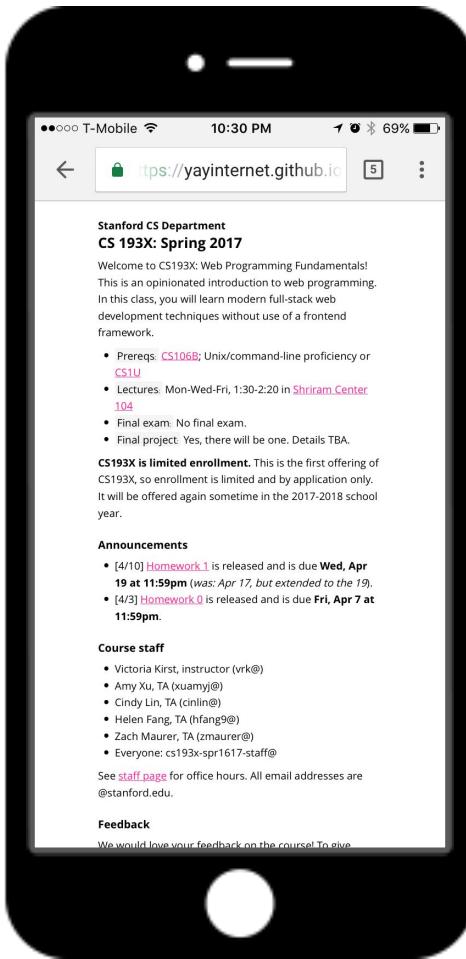
# Mobile web

# Say you have the following website:

The screenshot shows a web browser window with the following details:

- Title Bar:** CS 193X: Spring 2017
- Address Bar:** https://yayinternet.github.io/lecture06/mobile.html
- Page Content:**
  - Section Header:** Stanford CS Department  
**Section Title:** CS 193X: Spring 2017
  - Welcome Text:** Welcome to CS193X: Web Programming Fundamentals! This is an opinionated introduction to web programming. In this class, you will learn modern full-stack web development techniques without use of a frontend framework.
  - List:**
    - Prereqs: [CS106B](#); Unix/command-line proficiency or [CS1U](#)
    - Lectures: Mon-Wed-Fri, 1:30-2:20 in [Shriram Center 104](#)
    - Final exam: No final exam.
    - Final project: Yes, there will be one. Details TBA.
  - Note:** **CS193X is limited enrollment.** This is the first offering of CS193X, so enrollment is limited and by application only. It will be offered again sometime in the 2017-2018 school year.
  - Section Header:** Announcements
  - List:**
    - [4/10] [Homework 1](#) is released and is due **Wed, Apr 19 at 11:59pm** (*was: Apr 17, but extended to the 19*).
    - [4/3] [Homework 0](#) is released and is due **Fri, Apr 7 at 11:59pm**.
  - Section Header:** Course staff
  - List:**
    - Victoria Kirst, instructor (vrk@)
    - Amy Xu, TA (xuamyj@)
    - Cindy Lin, TA (cmlinlin@)
    - Helen Fang, TA (hfang9@)
    - Zach Maurer, TA (zmaurer@)
    - Everyone: cs193x-spr1617-staff@

**Q: What does it look like on a phone?**



Not terrible... but pretty small and hard to read.

# Responsive web design

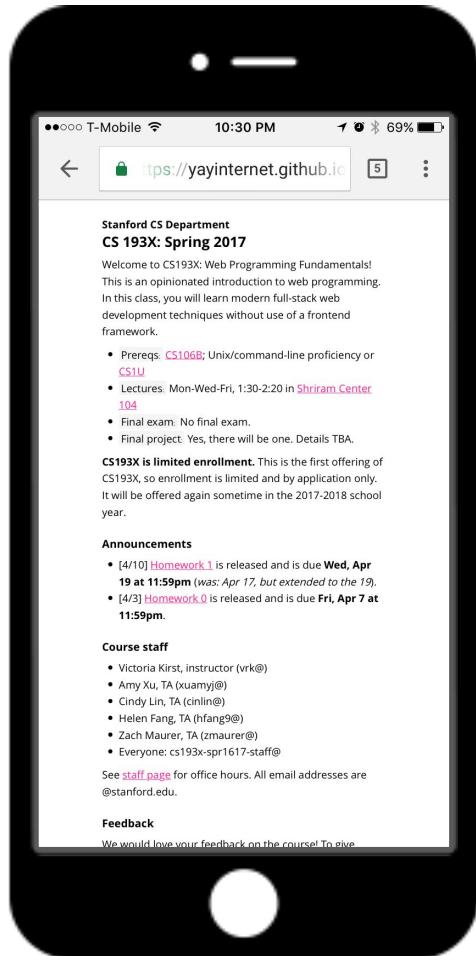
We want to write our CSS in a way that can look nice in a wide range of screen sizes:

- 27" desktop monitor
- Macbook Air
- Samsung Galaxy S7
- iPhone 7
- iPad

**Q: How do we do this?**

Do we need to write totally different  
CSS for every screen size?!

# Mobile sizing



Unless directed otherwise via HTML or CSS cues, mobile browsers render web pages at a **desktop screen width** (~1000px), then "zooms out" until the entire page fits on screen.

(That's why you sometimes get web pages with teeny-tiny font on your phone: these webpages have not added support for mobile.)

[\(Read more on how this works\)](#)

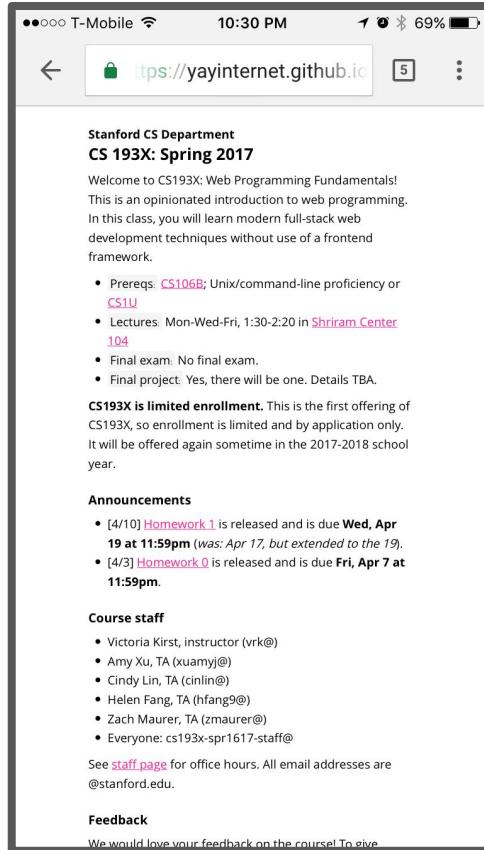
# Meta viewport tag

To prevent phone browsers from rendering the page at desktop width and zooming out, use the **meta viewport tag**:

```
<meta name="viewport"  
content="width=device-width, initial-scale=1">
```

This belongs in the `<head>` section of your HTML.  
(Same section as the `<title>`, `<link>`, and other metadata elements.)

# Meta viewport tag



**Without the meta  
viewport tag**



**With the meta  
viewport tag**

# Meta viewport tag

```
<meta name="viewport"  
content="width=device-width, initial-scale=1">
```

- **name=viewport**: "Browser, I am going to tell you how I want the viewport to look."
- **width=device-width**: "The viewport's width should always start at the device's width." (i.e., don't do that thing on mobile where you render the page at the desktop's width)
- **initial-scale=1**: "Start at zoom level of 100%."

# Meta viewport tag

```
<meta name="viewport"  
content="width=device-width, initial-scale=1">
```

**(You should pretty much always  
include this tag in your HTML.)**

# Making adjustments

The meta viewport tag gets us almost all the way there, but we want to make a few adjustments.

For example, the margin seems a tad too big on mobile. Can we set a different margin property for mobile?



# CSS media queries

You can define a **CSS media query** in order to change style rules based on the characteristics of the device:

```
@media (max-width: 500px) {  
    article {  
        margin: 0 2px;  
    }  
}
```

You can create [much more complex](#) media queries as well.



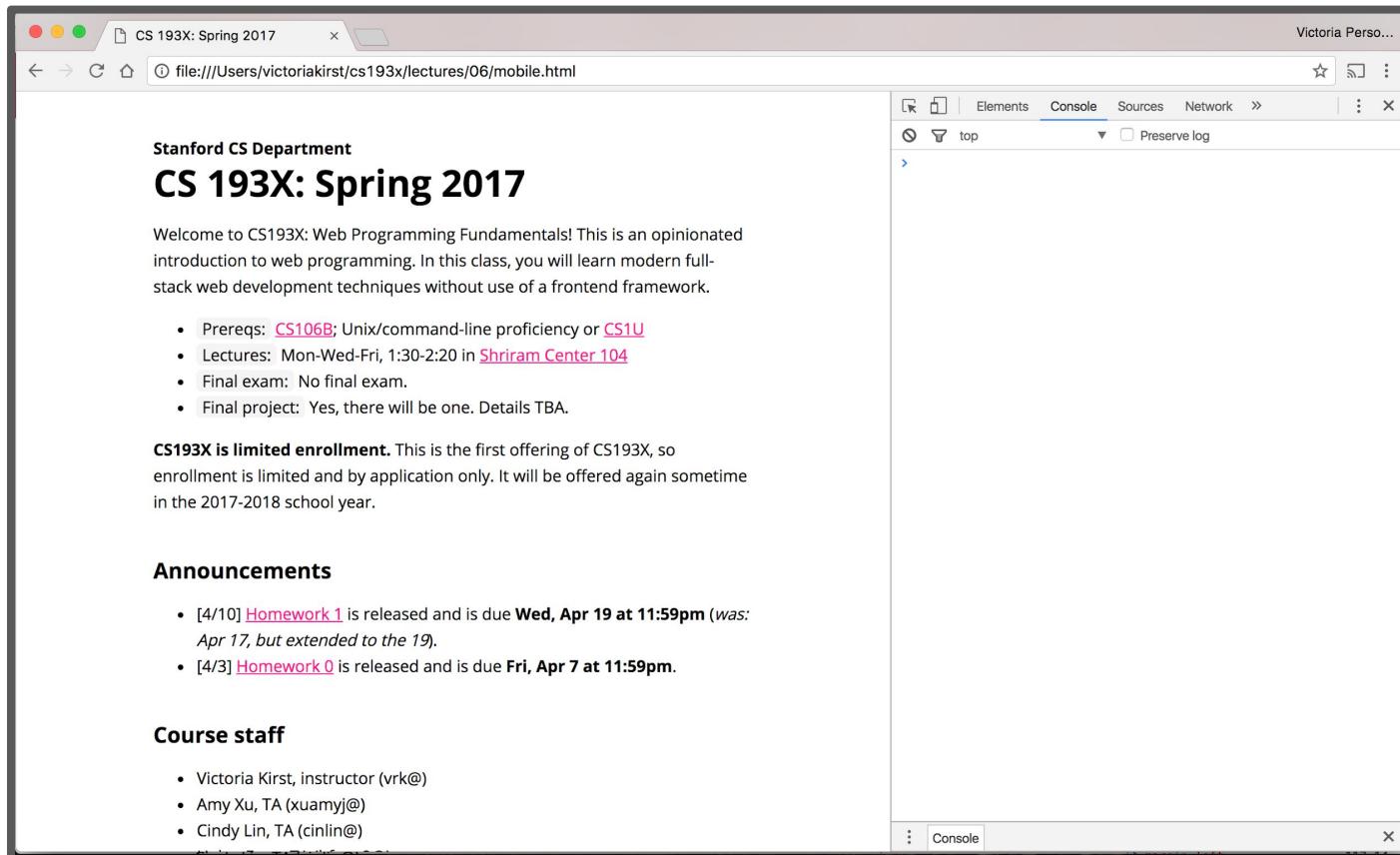
# Development strategies

Practical question: **How do you test mobile layouts?**

- Do you upload your HTML+CSS somewhere online and navigate to that URL on your phone?
- Is there a way to connect your phone to your local device?
- Do you run it in an Android/iOS emulator?
- Other?!

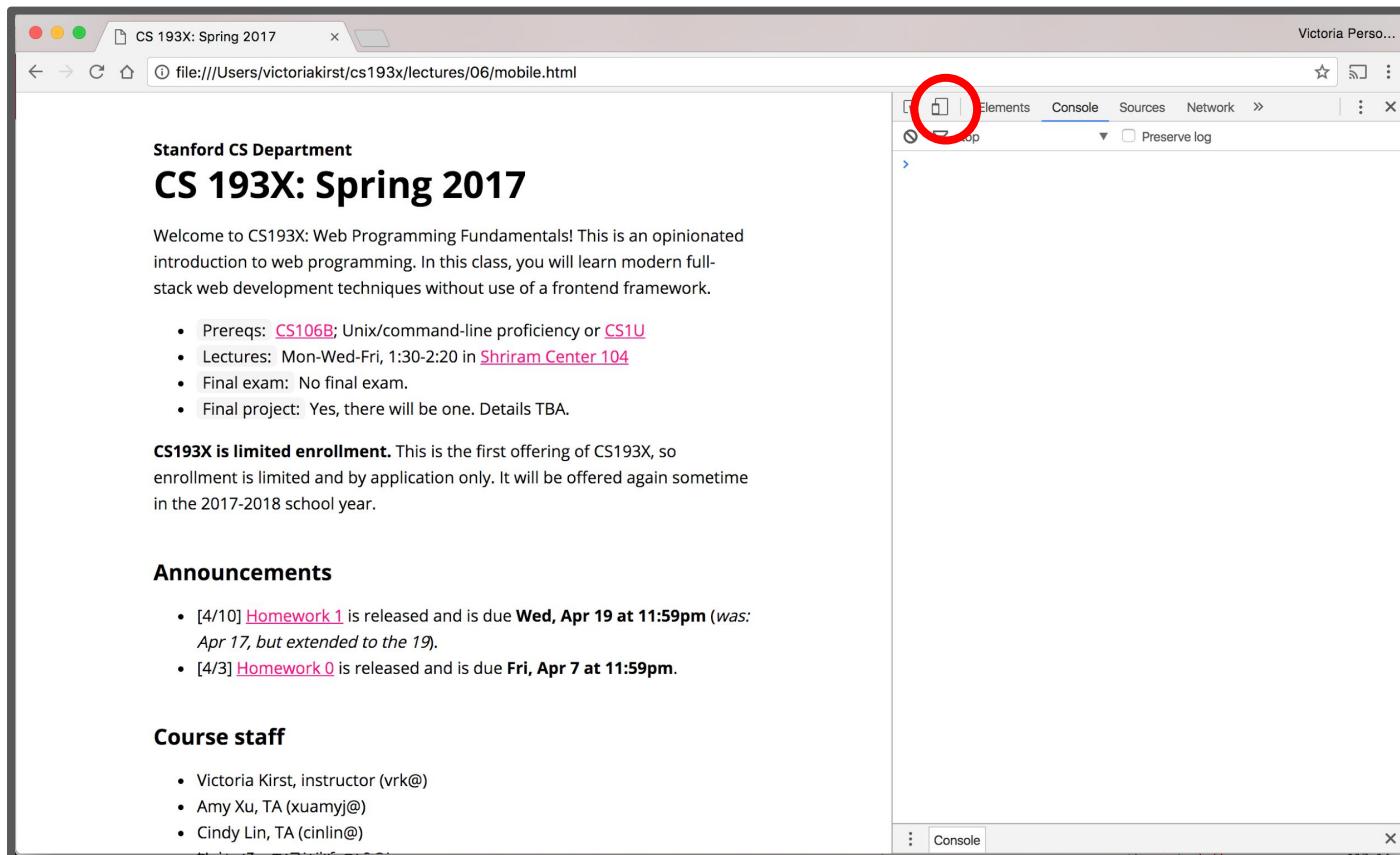
# Chrome device mode

You can simulate a web page in a mobile layout via [Chrome device mode](#):



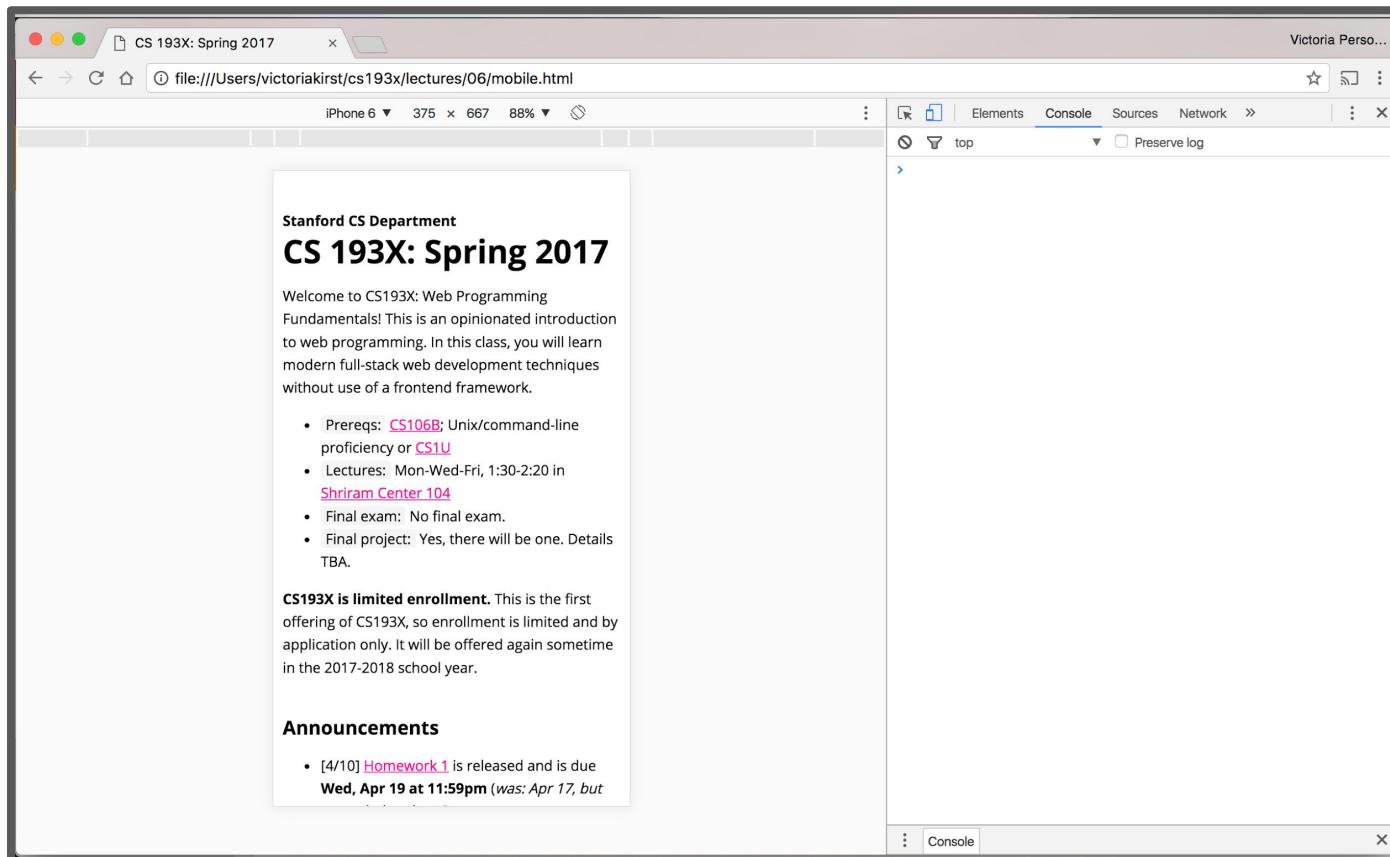
# Chrome device mode

You can simulate a web page in a mobile layout via [Chrome device mode](#):



# Chrome device mode

You can simulate a web page in a mobile layout via [Chrome device mode](#):



# Chrome device mode

## Advantages of Chrome device mode:

- Super convenient
- Mostly accurate

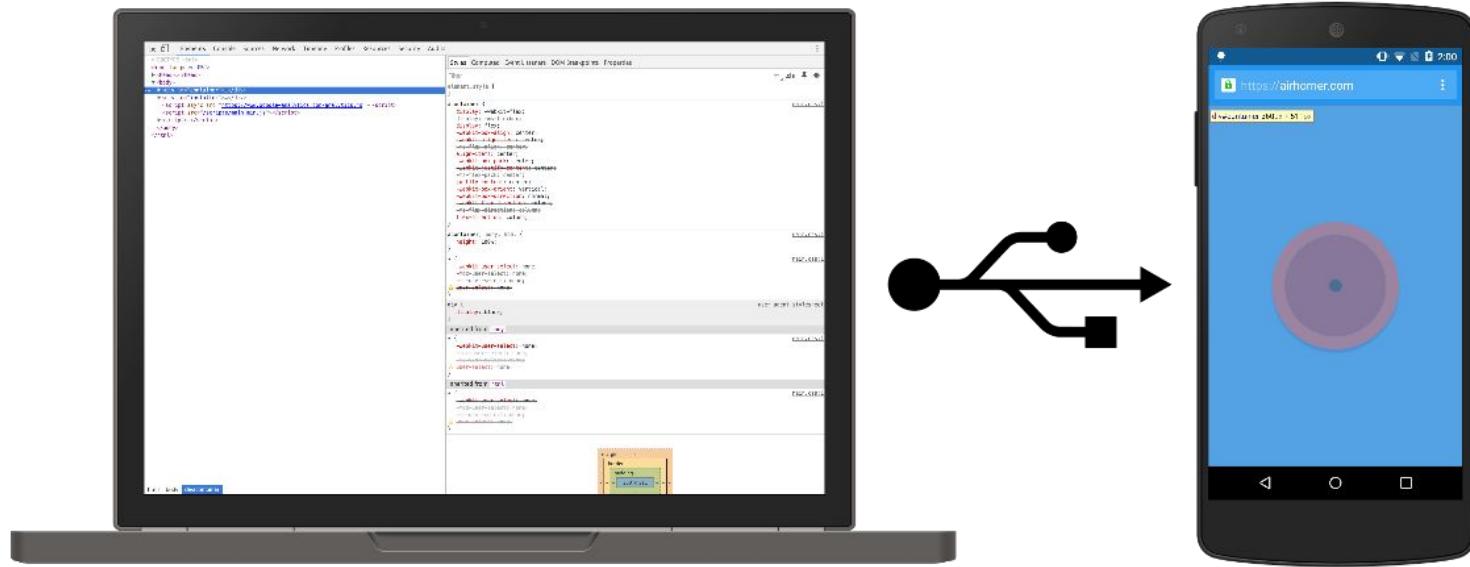
## Disadvantages of Chrome device mode:

- Not always accurate - iPhone particularly an issue
- A little buggy
- Doesn't simulate performance issues

You should always test on real devices, too.

# Chrome remote debugging

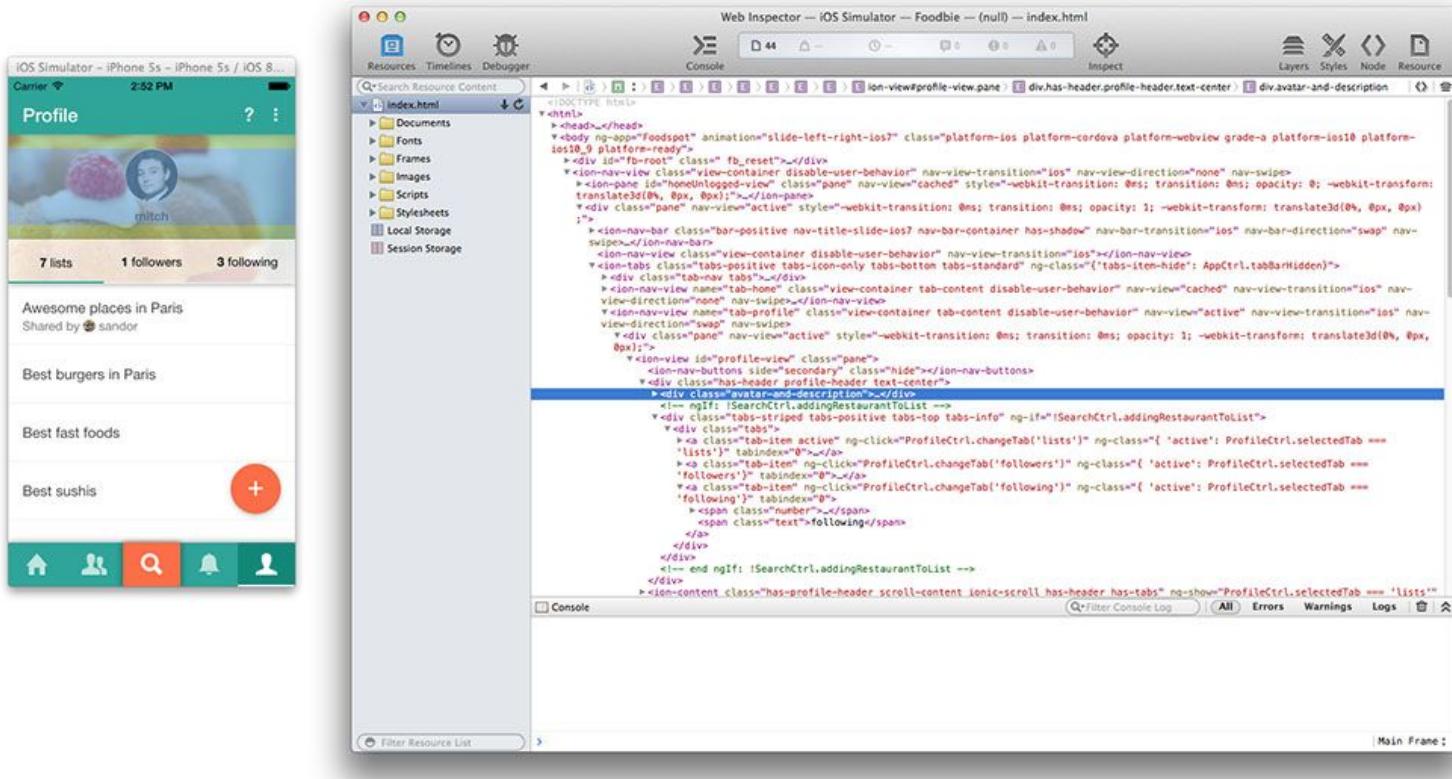
If you have an Android phone, you can debug web pages on your phone via [Chrome remote debugging](#).



(You can also load a server running locally on your laptop, on your phone via [port forwarding](#).  
But we haven't talked about servers yet.)

# Safari remote debugging

If you have an iPhone, you can debug web pages on your phone via Safari remote debugging.



**Relative font sizes:  
percent, em, rem**

# Relative units

Whenever possible, it's best to use **relative units** (like percentage) instead of absolute units (like px).

## Advantages:

- More likely to work on different screen sizes
- Easier to reason about; fewer magic numbers  
10% / 80% / 10% vs 122px / 926px / 122px

**Q: Should we be using relative units on font-size?**

# Relative font sizes: percent

You can define font sizes in terms of percentage:

```
<body>
  <h1>This is 60px</h1>
  <p>This is 15px</p>
</body>
```

```
body {
  font-size: 30px;
}

h1 {
  font-size: 200%;
}

p {
  font-size: 50%;
```

**This is 60px**

This is 15px

([CodePen](#))

# Relative font sizes: percent

Percent on font-size behaves exactly like percentage on width and height, in that it's relative to the parent:

```
<div>
  This is 60px
  <p>This is 45px</p>
</div>
```

```
body {
  font-size: 30px;
}

div {
  font-size: 200%;
}

p {
  font-size: 75%;
}
```

This is 60px

This is 45px

([CodePen](#))

# Relative font sizes: percent

Percent on font-size behaves exactly like percentage on width and height, in that it's relative to the parent:

```
<div>
  This is 60px
  <p>This is 45px</p>
</div>
```

```
body {
  font-size: 30px;
}

div {
  font-size: 200%;
}

p {
  font-size: 75%;
```

This is 60px

This is 45px

p is 75% of its parent, which  
is 200% of 30px.

p's size:  $.75 \times 2 \times 30 = 45\text{px}$

([CodePen](#))

# Relative font sizes: em

But instead of percentages, relative font sizes are usually defined in terms of em:

- em represents the calculated font-size of the element
  - 1em = the inherited font size
  - 2em = 2 times the inherited font size

In other words,

**font-size: 1em;** is the same as **font-size: 100%;**

# Relative font sizes: em

```
<body>
  <h1>This is 60px</h1>
  <p>This is 15px</p>
</body>
```

```
body {
  font-size: 30px;
}
```

```
div {
  font-size: 2em;
}
```

```
p {
  font-size: .5em;
}
```

**This is 60px**

This is 15px

([CodePen](#))

# Relative font sizes: em

```
<div>  
  This is 60px  
  <p>This is 45px</p>  
</div>
```

```
body {  
  font-size: 30px;  
}  
  
div {  
  font-size: 2em;  
}  
  
p {  
  font-size: .75em;  
}
```

This is 60px

This is 45px

([CodePen](#))

# Relative font sizes: em

```
<div>  
  This is 60px  
  <p>This is 45px</p>  
</div>
```

```
body {  
  font-size: 30px;  
}  
  
div {  
  font-size: 2em;  
}  
  
p {  
  font-size: .75em;  
}
```

This is 60px

This is 45px

p's inherited font size is 2em, which is 60px. So 0.75em is  $0.75 \times 60 = 45\text{px}$ .

([CodePen](#))

```
<body>  
  This is  
  <h1>  
    <strong>120px</strong>  
  </h1>  
</body>
```

```
body {  
  font-size: 30px;  
}  
  
strong {  
  font-size: 2em;  
}
```

This is  
**120px**

Wait, why is `<strong>` 120px and not 60px?

```
<body>  
  This is  
  <h1>  
    <strong>120px</strong>  
  </h1>  
</body>
```

```
body {  
  font-size: 30px;  
}  
  
strong {  
  font-size: 2em;  
}
```

This is  
**120px**

([CodePen](#))

```
h1 {  
  display: block;  
  font-size: 2em;  
  -webkit-margin-before: 0.67em;  
  -webkit-margin-after: 0.67em;  
  -webkit-margin-start: 0px;  
  -webkit-margin-end: 0px;  
  font-weight: bold;  
}
```

user agent stylesheet

In the Chrome Inspector, we see the default browser font-size for h1 is 2em. So it's  $30 * 2 * 2 = 120\text{px}$ .

# Relative font sizes: `rem`

If you **do not** want your relative font sizes to compound through inheritance, use `rem`:

- `rem` represents the `font-size` of the `root` element
  - $1\text{rem}$  = the root (`html` tag) font size
  - $2\text{rem}$  = 2 times root font size

# Relative font sizes: rem

```
<body>
  <div>
    This is 60px
    <p>This is 22.5px</p>
  </div>
</body>
```

```
html {
  font-size: 30px;
}
```

```
div {
  font-size: 2rem;
}

p {
  font-size: .75rem;
}
```

This is 60px

This is 22.5px

([CodePen](#))

# Relative font sizes: rem

```
<body>
  <div>
    This is 60px
    <p>This is 22.5px</p>
  </div>
</body>
```

```
html {
  font-size: 30px;
}
```

```
div {
  font-size: 2rem;
}
```

```
p {
  font-size: .75rem;
}
```

This is 60px

This is 22.5px

font-size is set on the  
html element, not body (or  
any other tag)

([CodePen](#))

# Relative font sizes: rem

```
<body>
  <div>
    This is 60px
    <p>This is 22.5px</p>
  </div>
</body>
```

This is 60px

This is 22.5px

```
html {
  font-size: 30px;
}
```

```
div {
  font-size: 2rem;
}
```

```
p {
  font-size: .75rem;
}
```

.75em is calculated from the root, which is 30px, so  $30 * .75 = 22.5\text{px}$ .

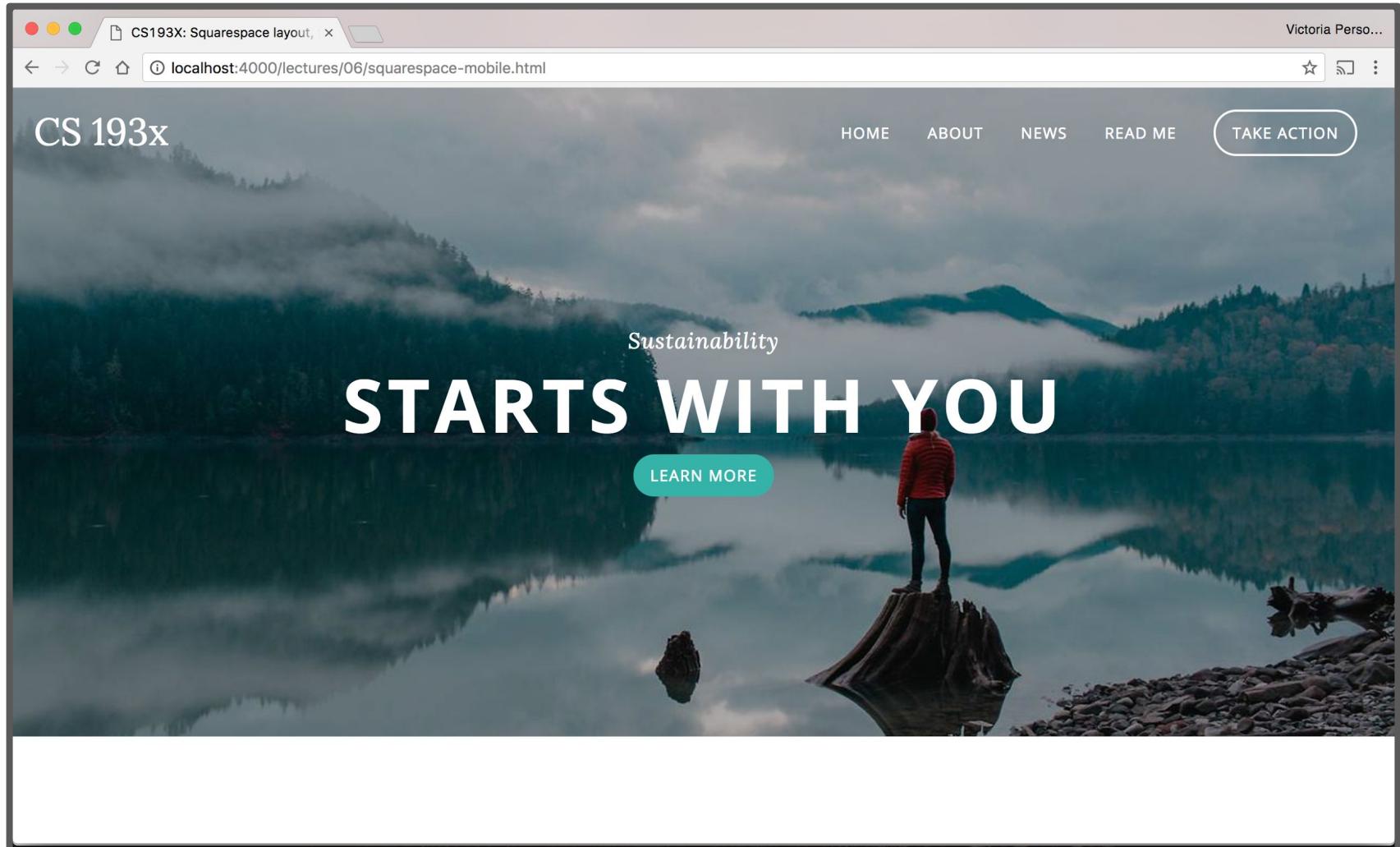
([CodePen](#))

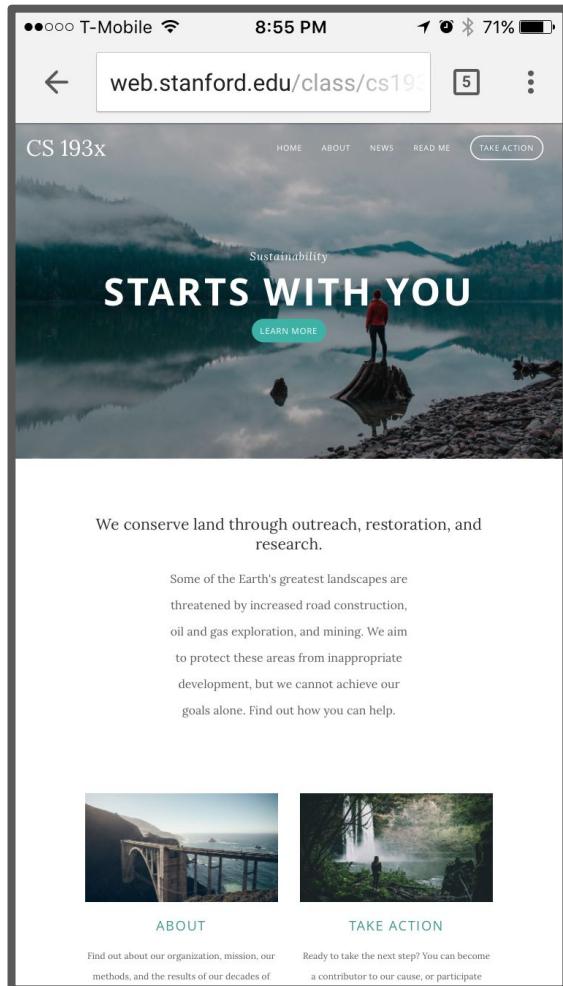
# Relative font conclusions

Use relative fonts for the same purpose as using relative heights and widths:

- Prefer em and rem over percentages
  - Not for any deep reason, but em is meant for font so it's semantically cleaner
- Use rem to avoid compounding sizes
- In addition to font-size, consider em/rem for:
  - line-height
  - margin-top
  - margin-bottom

What does our Squarespace layout look like in a phone  
with the meta viewport tag?





**Without** the meta  
viewport tag



**With** the meta  
viewport tag

The mobile layout features a dark header with "CS 193x" and a menu icon. Below is a large image of a person standing on a log in a misty, mountainous landscape. Text reads "Sustainability STARTS WITH YOU" with a "LEARN MORE" button. A white box contains text about land conservation and a message about protecting landscapes from development.

CS 193x

Sustainability  
**STARTS WITH YOU**

LEARN MORE

We conserve land through outreach, restoration, and research.

Some of the Earth's greatest landscapes are threatened by increased road construction, oil and gas exploration, and mining. We aim to protect these areas from inappropriate development, but we cannot achieve our goals alone. Find out how you can help.

The mobile layout features a dark header with "CS 193x" and a menu icon. Below is a large image of a bridge over a coastal cliff. A text box discusses the organization's mission to protect landscapes. At the bottom is a photo of a waterfall.

protect these areas from inappropriate development, but we cannot achieve our goals alone. Find out how you can help.



**ABOUT**

Find out about our organization, mission, our methods, and the results of our decades of advocacy.



**Completed mobile layout**

# Mobile summary

- Always add the **meta viewport tag**
- Use **@media queries** to add styles for devices with certain characteristics, such as screen width
- Use the **Chrome Device Mode** to simulate mobile rendering on desktop
- For height and width, prefer percentages
- For fonts, prefer em and rem
- Try to **minimize dependent rules** (Changing the width of one container force you to change 15 other properties to look right)

More on [responsive web design](#)

# CSS wrap-up

Even though we're "done" with CSS, we will be using CSS all quarter in homework and examples.

Later this quarter:

- More flexbox patterns
  - More [practice with flexbox / game](#)
- CSS animations
- Possibly [grid](#)