

# CS193X: Web Programming Fundamentals

Spring 2017

Victoria Kirst  
([vrk@stanford.edu](mailto:vrk@stanford.edu))

# Today's schedule

## Schedule:

- HTML: Background and history
- Complex selectors
- Box model
- Debugging with Chrome Inspector
- **Case study:** Squarespace Layout (will continue into Monday)

# (Forgot to mention: Paths)

`img src`, `a href`, and `link href` can all take either **relative** or **absolute** paths to the resource:

- `<a href="about.html">About</a>`
- ``
- `<link rel="stylesheet" href="css/style.css"/>`

If you are unfamiliar with paths, check out the following:

- [Absolute vs relative paths](#)
- [Unix directories and file paths](#)
- If anything's still unclear, come to [office hours](#)!

# HTML: Background and History

# What HTML elements can I use?

Q: Instead of `<span class="highlight"></span>`,  
can I create a `<highlight>` element?

```
<p>  
  The homework is  
  <highlight>due Friday</highlight>.  
</p>
```

```
highlight {  
  background-color: yellow;  
}
```

Q: Does this  
even work?

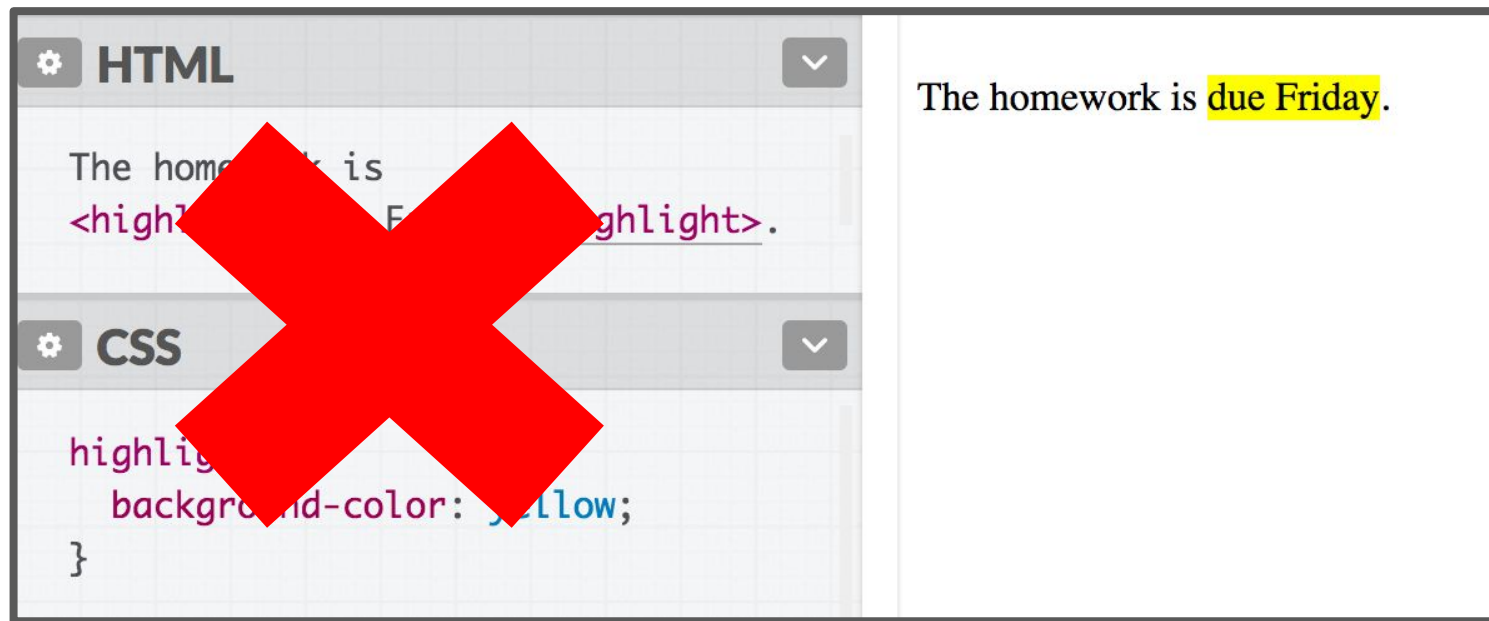
# What HTML elements can I use?

This renders correctly:



# What HTML elements can I use?

This renders correctly:

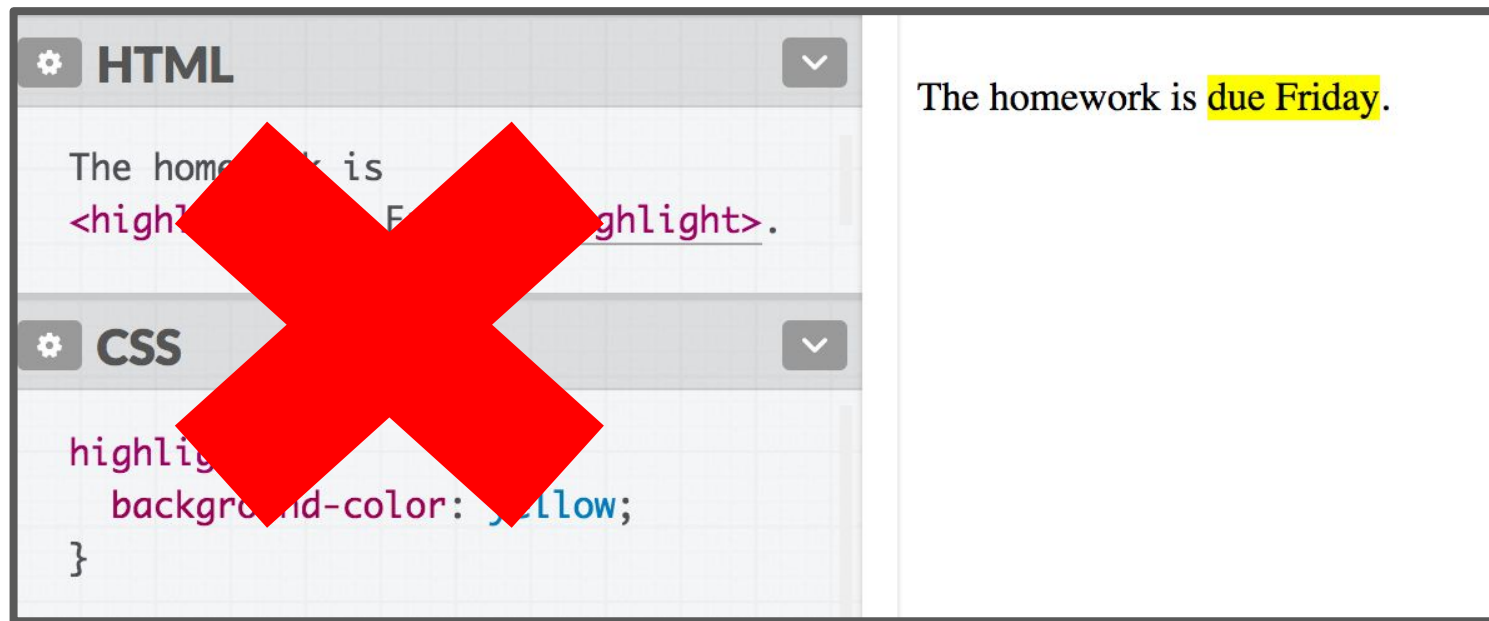


**But you shouldn't do this!**

It is non-standard behavior.

# What HTML elements can I use?

This renders correctly:



**What?!?!?**

**But you shouldn't do this!**  
It is non-standard behavior.



# What?!

- What is "standard" HTML?
- Why does invalid HTML/CSS still work sometimes?
  - If my Java code is wrong, I get a compiler error... If my HTML or CSS is wrong, why don't I get an error?
- Why does it matter that I follow "standard" HTML?

# A very brief history of HTML

# History



**Tim  
Berners-Lee**

- 1989: World Wide Web created  
(WWW: web pages and the protocol in which they are served HTTP/HTTPS)
- 1994: World Wide Web Consortium created
  - "**W3C**": Goal to maintain and develop standards about how the web should work
  - Oversees several languages:
    - HTML, CSS, DOM, XML, etc
- 1997: "HTML4" published
  - The first major stable version of HTML

# Degrading gracefully

The W3C HTML spec lists several [design principles](#), and one is degrading gracefully:



"An escalator  
can never  
break: it can  
only become  
stairs"

This is why browsers do a **best-effort** to render non-standard ("invalid") HTML and CSS.

# Best-effort rendering



It's also why `<highlight>` "works", even though it's Invalid HTML.

# Why not enforce strict HTML?

It's super weird that:

- Browsers don't fail when given invalid HTML / CSS
- Browsers not only don't fail, but they render invalid HTML/CSS seemingly "correctly"

**Q: Why doesn't the browser reject poorly written HTML/CSS?**

# Why not enforce strict HTML?

It's super weird that:

- Browsers don't fail when given invalid HTML / CSS
- Browsers not only don't fail, but they render invalid HTML/CSS seemingly "correctly"

Q: Why doesn't the browser reject poorly written HTML/CSS?

**A: There was a (failed) attempt to enforce this, but it was too late: the Internet grew too big!**

# The nerdy, mostly\* accurate backstory for HTML today

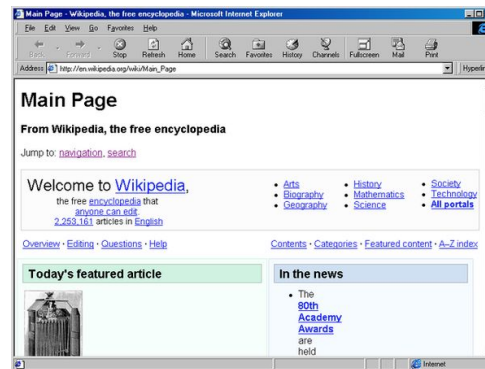
\*I would be more accurate, but it's hard to get valid sources online... so I'm going off of what I can + the lore I've heard while working on a browser.



# State of the world, 1997:



Standards say  
one thing,



Browsers do  
another thing,

Developers write  
weird, non-standard  
code.



# State of the world, 1997:

**In 1997, things are  
kind of a mess!**



Standard  
one thing,

another thing,

Developers write  
weird, non-standard  
code.



# 2000ish:



**W3C**

Oops, that "degrading gracefully" thing was a mistake.

Browsers, stop rendering pages that have invalid HTML.

**(This what the proposal of XHTML 1.1)**

# 2000ish: (not totally accurate)



W3C

**That would  
break the  
internet!**

**What?!?!**

Sure  
whatever



**We can't  
do that!**



# 2004: WHATWG formed



**Let's burn everything and  
start from scratch with  
XHTML 1.1**

(break approx. 64 million websites)



**WHATWG**



**Let's work on HTML5**  
(an imperfect but realistic standard)

# Fast forward 2017?!



- W3C gave up XHTML 1.1 in 2007
- W3C and WHATWG are mostly friends (I think), though they are still separate entities
- Can still find some snarky quotes on [WHATWG website](http://whatwg.org)

# "HTML5" vs HTML

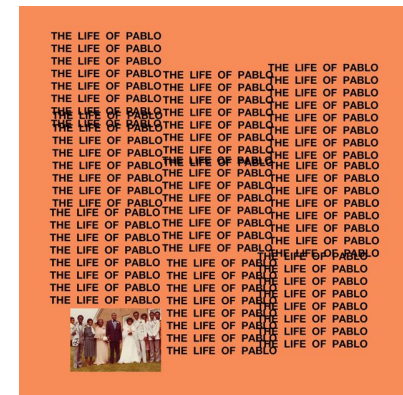
W3C maintains [HTML5](#):

- More stable version of WHATWG's HTML
- Usually copies what WHATWG does after the dust settles



WHATWG maintains [HTML: The Living Standard](#)

- No number, no versions
- Updated frequently and being updated today!
- Most browsers implement WHATWG
- This is why I don't say "HTML5"



# What you need to know

**Q: What HTML elements can I choose from?**

- Check [MDN's list of HTML tags](#)

**Q: How do I know if an HTML tag (or CSS property, or JS feature) is implemented on all browsers?**

- Check [caniuse.com](#)

**Q: Why shouldn't I use non-standard HTML/CSS/JavaScript, even if it works in every browser?**



# What you need to know

**Q: What HTML elements can I choose from?**

- Check [MDN's list of HTML tags](#)

**Q: How do I know if an HTML tag (or CSS property, or JS feature) is implemented on all browsers?**

- Check [caniuse.com](#)

**Q: Why shouldn't I use non-standard HTML/CSS/JavaScript, even if it works in every browser?**

- Because it won't be guaranteed to work in the future
- Because it won't be guaranteed to work on all "user agents" (not just browsers)

# What you need to know

## Q: Wouldn't it be super useful to create custom elements?

- Yes! There is a [spec for this](#) currently under development.
  - (Note that custom elements are not really meant for our example; custom elements are meant for defining custom behavior and not just style. For defining style, CSS classes/ids are still most appropriate.)

Back to writing  
code!

# CSS Selectors: Classes and Ids

# Classes and ids

There are 3 basic types of CSS selectors:

Element selector (this is the one we've been using)	<b>p</b>	All <b>&lt;p&gt;</b> elements
❖ ID selector ❖	<b>#abc</b>	element with <b>id="abc"</b>
❖ Class selector ❖	<b>.abc</b>	elements with <b>class="abc"</b>

```
<h1 id="title">Homework</h1>
<em class="hw">HW0</em> is due Friday.<br/>
<em class="hw">HW1</em> goes out Monday.<br/>
<em>All homework due at 11:59pm.</em>
```

Other selectors

# *element.className*

Syntax	Example	Example described
<i>element.className</i>	p.abc	<p> elements with abc class

HTML

```
<h1 class="hw">Homework 0</h1>
<p class="hw">Due Fri</p>
<p class="hw">Late cutoff Sun</p>
<h1>Lectures</h1>
<p>Apr 3: Syllabus</p>
<p>Apr 5: HTML+CSS</p>
```

CSS

```
p.hw {
  color: green;
}
```

## Homework 0

Due Fri

Late cutoff Sun

## Lectures

Apr 3: Syllabus

Apr 5: HTML+CSS

# Descendent selector

Syntax	Example	Example described
<i><b>selector selector</b></i>	<b>div strong</b>	<b>&lt;strong&gt;</b> elements that are descendants of a <b>&lt;div&gt;</b>

HTML

```
<div class="hw">
  <h1>Homework 0</h1>
  <p>Due Fri</p>
  <p>Late cutoff Sun</p>
</div>
```

CSS

```
.hw p {
  color: green;
}
```

## Homework 0

Due Fri

Late cutoff Sun

## Lectures

Apr 3: Syllabus

Apr 5: HTML+CSS



# Descendent selector

Syntax	Example	Example described
<i><b>selector selector</b></i>	<b>div strong</b>	<b>&lt;strong&gt;</b> elements that are descendants of a <b>&lt;div&gt;</b>

**Note:** The element does not have to be a direct child. The descendent may be nested many layers in.

The screenshot displays a web development interface with three panels. The top panel, titled 'HTML', contains the following code:

```
<div class="hw">
  <div>
    <p>
      HW0: <strong>Due Friday</strong>
    </p>
  </div>
  HW1 out <strong>Monday</strong>
</div>
```

The middle panel, titled 'CSS', contains the following code:

```
.hw strong {
  color: red;
}
```

The right panel shows the rendered output:

HW0: **Due Friday**

HW1 out **Monday**

# Descendent selector

Syntax	Example	Example described
<i><b>selector selector</b></i>	<b>div strong</b>	<b>&lt;strong&gt;</b> elements that are descendants of a <b>&lt;div&gt;</b>

## Discouraged:

```
<h1 class="hw">Homework 0</h1>  
<p class="hw">Due Fri</p>  
<p class="hw">Late cutoff Sun</p>
```

vs

## Preferred:

```
<div class="hw">  
  <h1>Homework 0</h1>  
  <p>Due Fri</p>  
  <p>Late cutoff Sun</p>  
</div>
```

Instead of applying a class to several adjacent elements, wrap the group in a `<div>` container and style the contents via descendent selectors.

# selector, selector (comma)

Syntax	Example	Example described
<i>selector, selector</i>	<b>h2, div</b>	<h2> elements and <div>s

HTML

```
<h1>Course Info</h1>
<h2>Lectures</h2>
<p>Mon-Wed-Fri 1:30-2:20</p>
<h2>Honor code</h2>
<p>Do the right thing</p>
```

CSS

```
h1, h2 {
  font-family: Arial;
}
```

## Course Info

### Lectures

Mon-Wed-Fri 1:30-2:20

### Honor code

Do the right thing

# Selector summary

Example	Example described
<b>p</b>	All <b>&lt;p&gt;</b> elements
<b>.abc</b>	All elements with the <b>abc class</b> , i.e. <b>class="abc"</b>
<b>#abc</b>	Element with the <b>abc id</b> , i.e. <b>id="abc"</b>
<b>p.abc</b>	<b>&lt;p&gt;</b> elements with <b>abc class</b>
<b>p#abc</b>	<b>&lt;p&gt;</b> element with <b>abc id</b> ( <b>p</b> is redundant)
<b>div strong</b>	<b>&lt;strong&gt;</b> elements that are descendants of a <b>&lt;div&gt;</b>
<b>h2, div</b>	<b>&lt;h2&gt;</b> elements and <b>&lt;div&gt;</b> s

# Grouping selectors

## 2 Common bugs:

p.abc    **vs**    p .abc

p .abc    **vs**    p, .abc

- A <p> element with the **abc** class **vs**  
An element with the **abc** class that descends from <p>
- An element with the **abc** class that descends from <p> **vs**  
All <p> elements *and* all elements with the **abc** class

# Combining selectors

You can combine selectors:

```
#main li.important strong {  
  color: red;  
}
```

**Q: What does this select?**

# Grouping selectors

**Q: What does this select?**

```
#main li.important strong {  
  color: red;  
}
```

**A: Read from right to left:**

- `<strong>` tags that are children of `<li>` tags that have an "important" class that are children of the element with the "main" id.

# Colliding styles

When styles collide, the most specific rule wins ([specificity](#))

```
div strong { color: red; }  
strong { color: blue; }
```

```
<div>  
  <strong>What color am I?</strong>  
</div>
```



# Colliding styles

When styles collide, the most specific rule wins ([specificity](#))

```
div strong { color: red; }  
strong { color: blue; }
```

```
<div>  
  <strong>What color am I?</strong>  
</div>
```

# Colliding styles

Specificity precedence rules ([details](#)):

- `ids` are more specific than `classes`
- `classes` are more specific than element names
- elements are more specific than children of those elements

# Colliding styles

- If elements have the same specificity, the later rule wins.

```
strong { color: red; }  
strong { color: blue; }
```

```
<div>  
  <strong>What color am I?</strong>  
</div>
```

Aside: The process of figuring out what rule applies to a given element is called the [cascade](#). This is where the "C" in *Cascading* Style Sheets comes from.

# Inheritance

We saw earlier that CSS styles are inherited from parent to child.

Instead of selecting all elements individually:

```
a, h1, p, strong {  
    font-family: Helvetica;  
}
```

You can style the parent and the children will inherit the styles.

```
body {  
    font-family: Helvetica;  
}
```

You can override this style via specificity:

```
h1, h2 {  
    font-family: Consolas;  
}
```

# Inheritance

While many CSS styles are inherited from parent to child,  
**not all CSS properties are inherited.**

```
a {  
  display: block;  
  font-family: Arial;  
}
```

<em> inherits the  
font-family property,  
but not display:

```
<a href="/home">  
  Back to <em>Home</em>  
</a>
```

[Back to Home](#)

# Inheritance

While many CSS styles are inherited from parent to child, **not all CSS properties are inherited.**

- There's no rule for what properties are inherited or not; the inheritance behavior defined in the CSS spec.
- You can look it up via MDN, e.g.

[font-family](#): Inherited yes

[display](#): Inherited no

- Generally text-related properties are inherited and layout-related properties are not.
- (You can also change this via the [inherit](#) CSS property, which is somewhat esoteric and not often use)

Before we move on:  
A few style notes

# Why not `<div>` everywhere?

Technically, you can define your entire web page using `<div>` and the `class` attribute.

- Is this a good idea?
- Why does HTML have `ids` when you have `classes`?
- Why does HTML have `<p>`, `<h1>`, `<strong>`, etc. when you have `<div>`, `<span>`, `class`, and `id`?

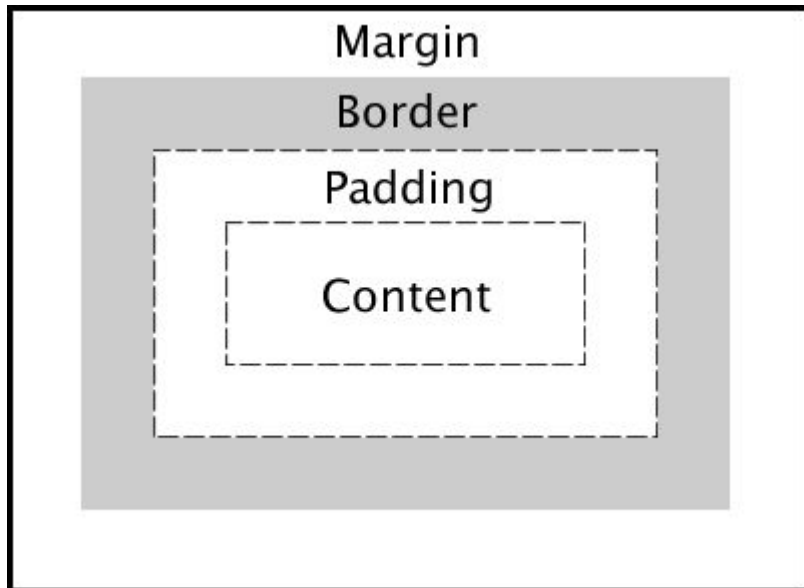


# CSS Box Model

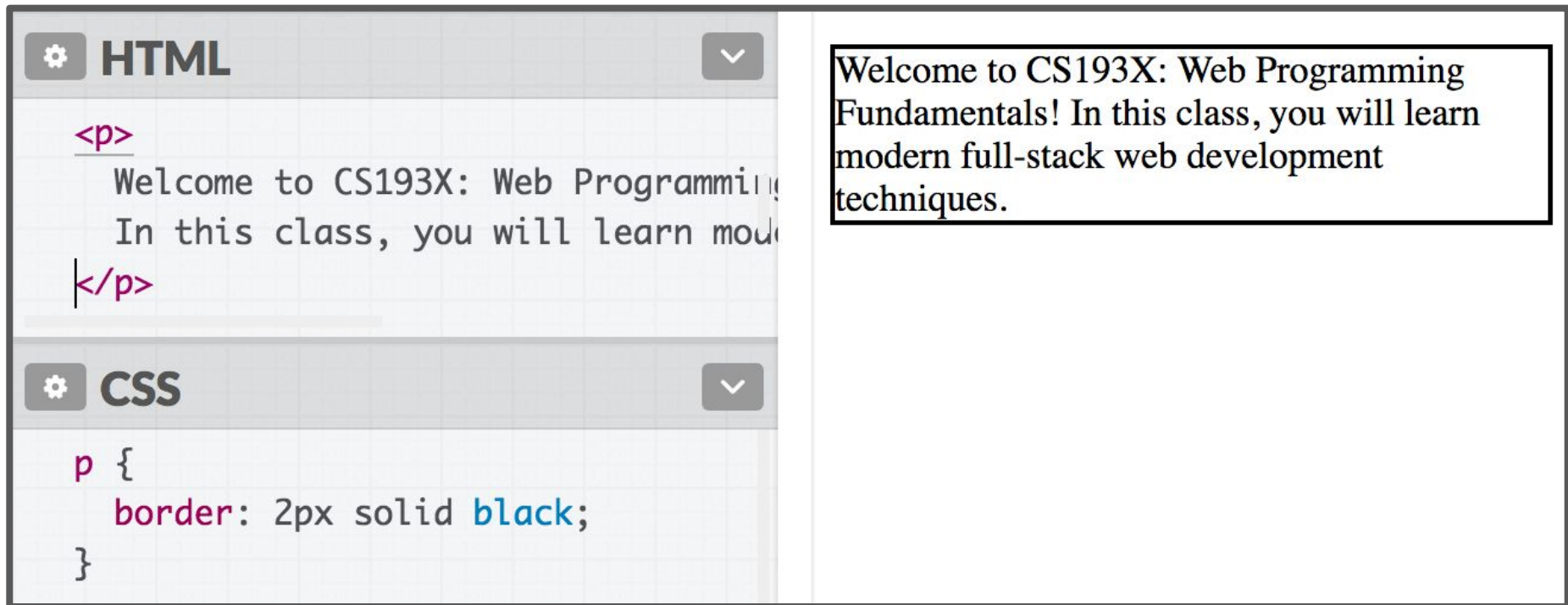
# The CSS Box Model

Every element is composed of 4 layers:

- the element's content
- the **border** around the element's content
- **padding** space between the content and border (inside)
- a **margin** clears the area around border (outside)



# border



We've used the [shorthand](#):  
`border: width style color;`

# border

Can also specify each border individually:

`border-top`

`border-bottom`

`border-left`

`border-right`

And can set each property individually:

`border-style: dotted;      (all styles)`

`border-width: 3px;`

`border-color: purple;`

# border

Can also specify each border individually:

`border-top`

`border-bottom`

`border-left`

`border-right`

And can set each property individually:

`border-style: dotted;` ([all styles](#))

`border-width: 3px;`

`border-color: purple;`

There are other units besides pixels (px) but we will address them in the next couple lectures.

# Rounded border

Can specify the `border-radius` to make rounded corners:

```
border-radius: 10px;
```

You don't actually need to set a border to use `border-radius`.

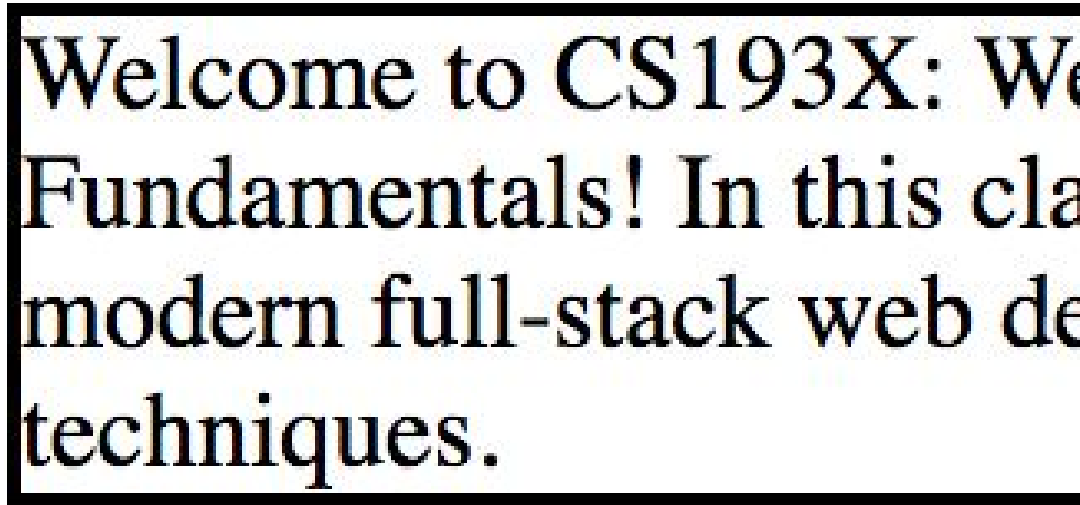
```
p {  
  background-color: purple;  
  border-radius: 10px;  
  color: white;  
}
```

Welcome to CS193X: Web Programming Fundamentals! In this class, you will learn modern full-stack web development techniques.

# Borders look a little squished

When we add a border to an element, it sits flush against the text:

**Q: How do we add space between the border and the content of the element?**



Welcome to CS193X: Web Fundamentals! In this class, we'll learn modern full-stack web development techniques.

# padding

```
p {  
  border: 2px solid black;  
  padding: 10px;  
}
```

Welcome to CS193X: Web Programming Fundamentals! In this class, you will learn modern full-stack web development techniques.

padding is the space between the border and the content.

- Can specify padding-top, padding-bottom, padding-left, padding-right
- There's also a shorthand:

padding: 2px 4px 3px 1px; <- top | left | bottom | right

padding: 10px 2px; <- top+bottom | left+right



# <div>s look a little squished

When we add a border to multiple divs, they sit flush against each other:



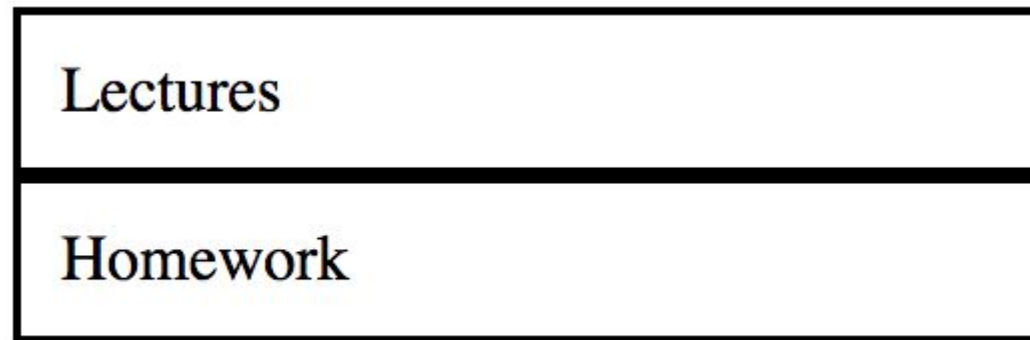
The screenshot shows a code editor with two panels. The left panel, titled 'HTML', contains the following code:

```
<div>
  Lectures
</div>
<div>
  Homework
</div>
```

The right panel, titled 'CSS', contains the following code:

```
div {
  border: 2px solid black;
  padding: 10px;
}
```

**Q: How do we add space between multiple elements?**



# margin

```
div {  
  margin: 20px;  
  padding: 10px;  
  border: 2px solid black;  
}
```

Lectures

Homework

margin is the space between the border and other elements.

- Can specify margin-top, margin-bottom, margin-left, margin-right
- There's also a shorthand:

margin: 2px 4px 3px 1px; <- top | left | bottom | right

margin: 10px 2px; <- top+bottom | left+right

# margin

Actually, why doesn't this:

```
div {  
  margin: 20px;  
  padding: 10px;  
  border: 2px solid black;  
}
```

Lectures

Homework

Look more like this?

Lectures

Homework

# margin

Actually, why doesn't this:

```
div {  
  margin: 20px;  
  padding: 10px;  
  border: 2px solid black;  
}
```

Lectures

Homework

...look more like this?

**20px margin-bottom** +  
**20px margin top** =  
40px margin?

Lectures

Homework

# margin collapsing

Sometimes the top and bottom margins of block elements are combined ("collapsed") into a single margin.

- This is called **margin collapsing**

Generally if:

- The elements are siblings
- The elements are block-level (not inline-block)

---

Lectures

Homework

Syllabus

then they collapse into **max**(*Bottom Margin, Top Margin*).

(There are [some exceptions](#) to this, but when in doubt, use the Page Inspector to see what's going on.)

# Box model for inline elements?

Q: Does the box model apply to inline elements as well?

# Box model for inline elements?

Q: Does the box model apply to inline elements as well?

A: Yes, but the box is shaped differently.

```

CSS

strong {
  border: 3px solid hotpink;
  padding: 5px;
  margin: 25px;

  background-color: lavenderblush;
}

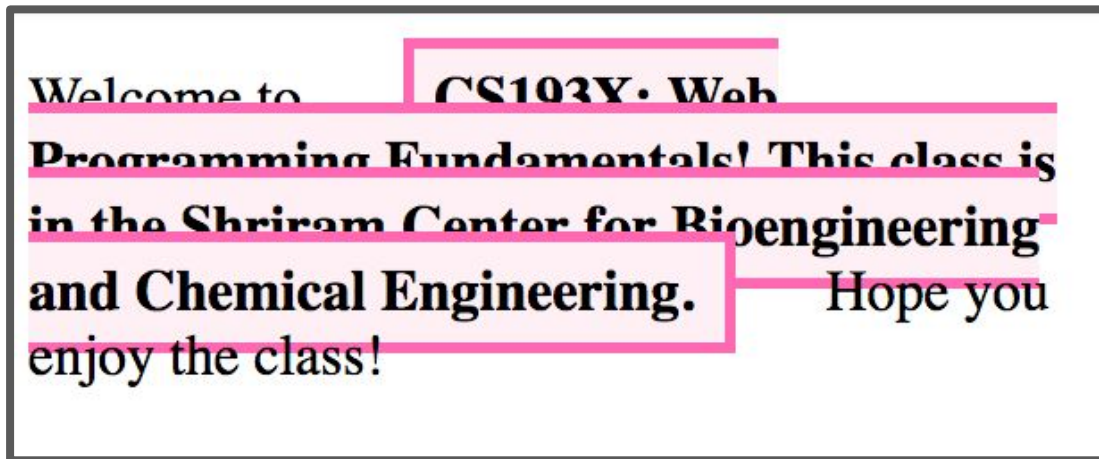
```

```

HTML

<p>
  Welcome to
  <strong>
    CS193X: Web Programming I
  </strong>
  Hope you enjoy the class!
</p>

```



# Box model for inline elements?

Q: Does the box model apply to inline elements as well?

A: Yes, but the box is shaped differently.

```

CSS

strong {
  border: 3px solid hotpink;
  padding: 5px;
  margin: 25px;

  background-color: lavenderblush;
}

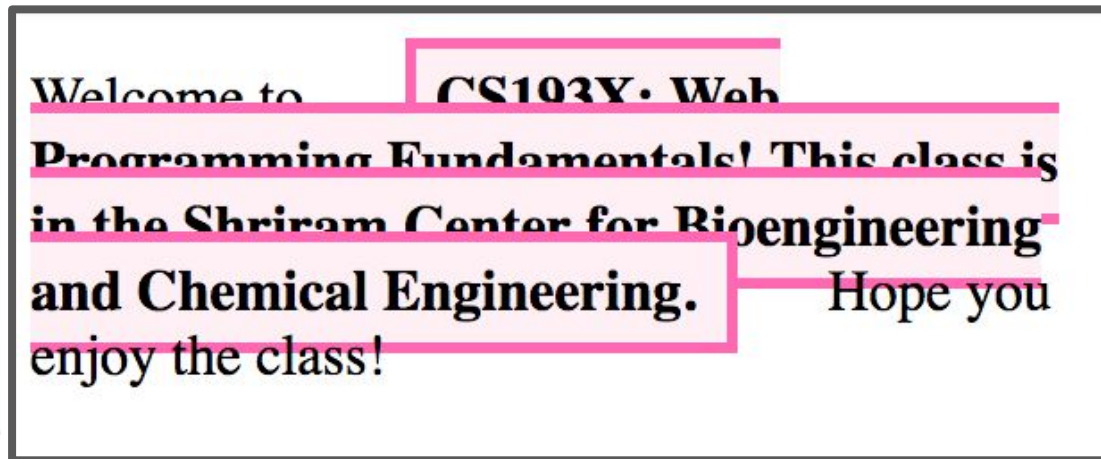
```

```

HTML

<p>
  Welcome to
  <strong>
    CS193X: Web Programming
  </strong>
  Hope you enjoy the class!
</p>

```



Let's change the line  
height to view this more  
clearly...



# Inline element box model

```

CSS

strong {
  border: 3px solid hotpink;
  padding: 5px;
  margin: 25px;
  line-height: 50px;
  background-color: lavenderblush;
}

```

Welcome to

**CS193X: Web**

**Programming Fundamentals! This class is**

**in the Shriram Center for Bioengineering**

**and Chemical Engineering.**

Hope you

enjoy the class!

([Codepen](#))

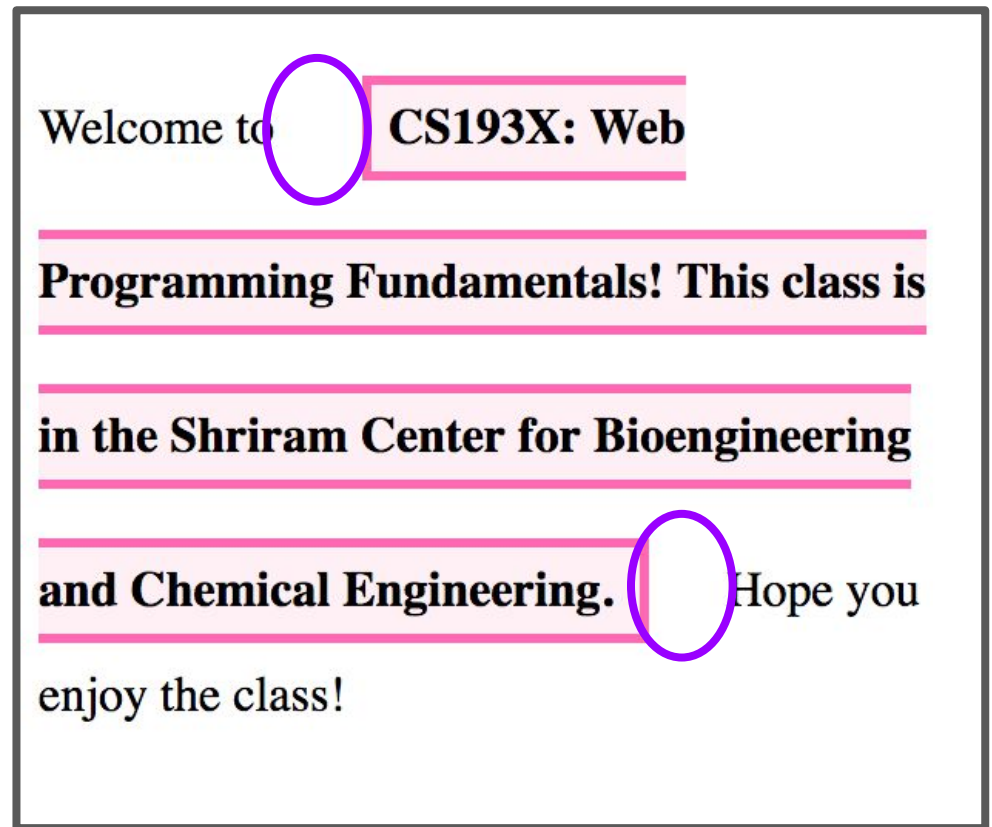
# Inline element box model

```

- CSS
strong {
  border: 3px solid hotpink;
  padding: 5px;
  margin: 25px;
  line-height: 50px;|
  background-color: lavenderblush;
}

```

- **margin** is to the left and right of the inline element
  - margin-top and margin-bottom are ignored
- use **line-height** to manage space between lines



([Codepen](#))

# The CSS Box Model

Let's revisit our Course web page example:

## CS 193X: Web Fun

### Announcements

4/3: Homework 0 is out! Due Friday.

4/3: Office hours are now posted.

[View Syllabus](#)

**Q: What does  
this look like in  
the browser?**

```
div {  
  display: inline-block;  
  background-color: yellow;  
}
```

```
<body>  
  <div>  
    <p>Make the background color yellow!</p>  
    <p>Surrounding these paragraphs</p>  
  </div>  
</body>
```

Make the background color yellow!

Surrounding these paragraphs

**Q: Why is there a  
white space  
around the box?**

We can use the  
browser's Page  
Inspector to help us  
figure it out!

# body has a default margin

Set `body { margin: 0; }` to make your elements lay flush to the page.

```
body {  
  margin: 0;  
}  
  
div {  
  display: inline-block;  
  background-color: yellow;  
}
```

Make the background color yellow!

Surrounding these paragraphs

# Recap so far...

We've talked about:

- **block vs inline** and the "natural" layout of the page, depending on the element type
- **classes and ids** and how to specify specific elements and groups of elements
- **div and span** and how to create generic elements
- **The CSS box model** and how every element is shaped like a box, with content -> padding -> border -> margin

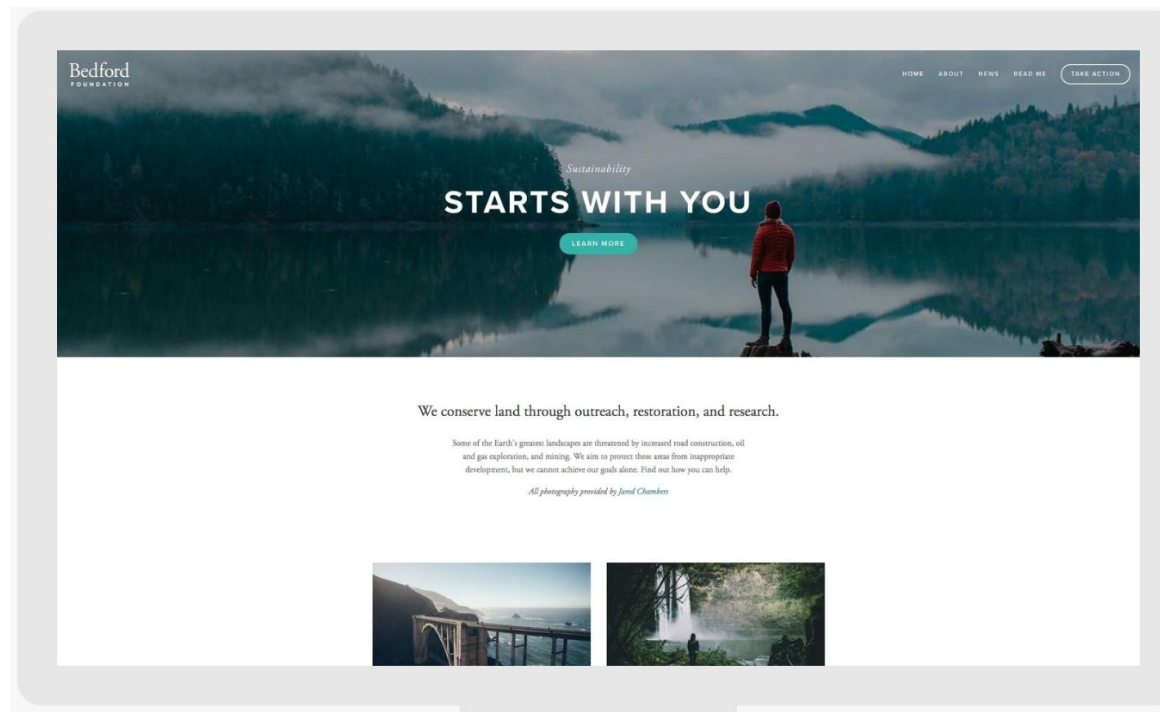
**Let's try making a "real" looking page!**

# Layout exercise



# Squarespace template

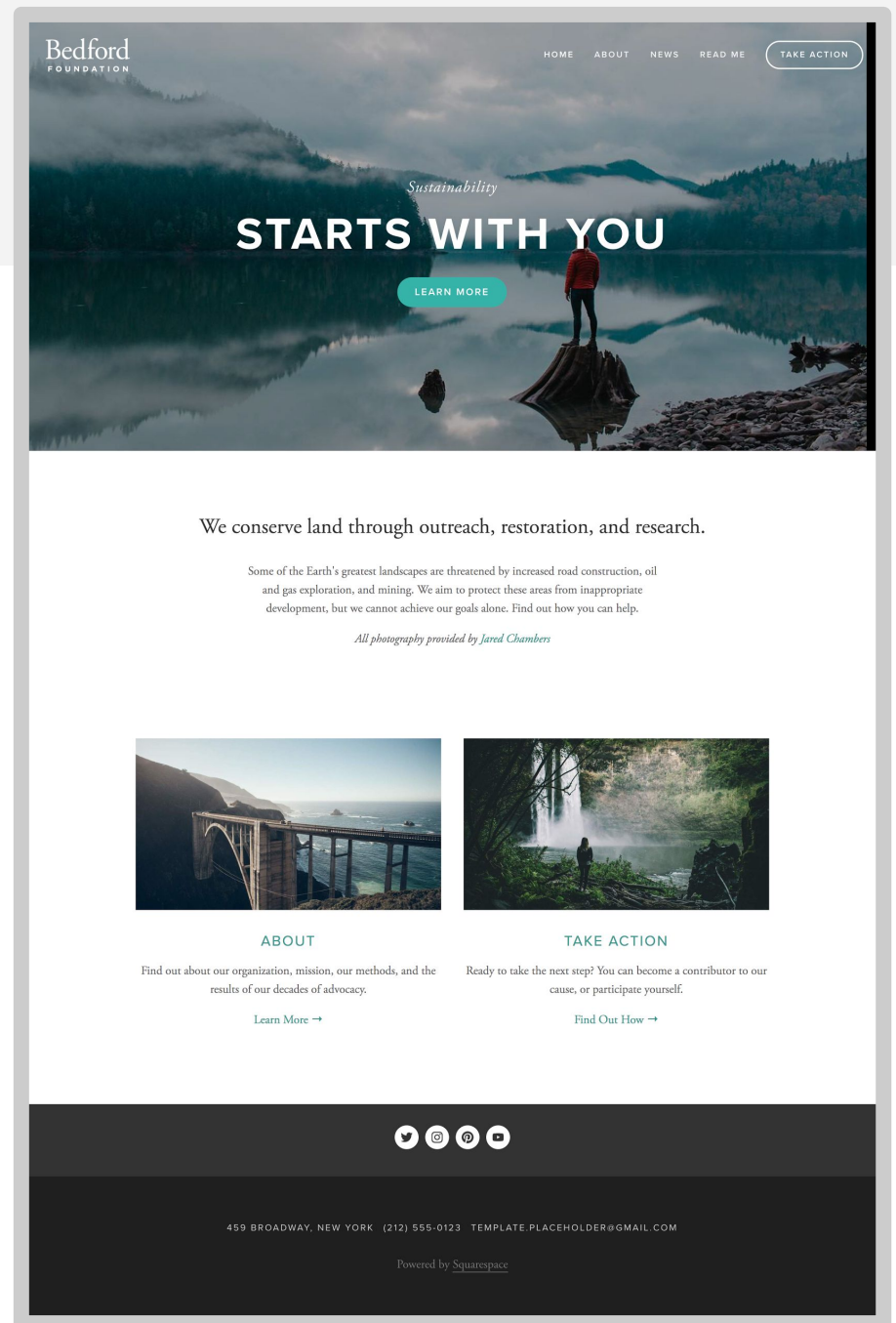
[Squarespace](#)'s most popular template looks like [this](#):



Do we know enough to make something like that?

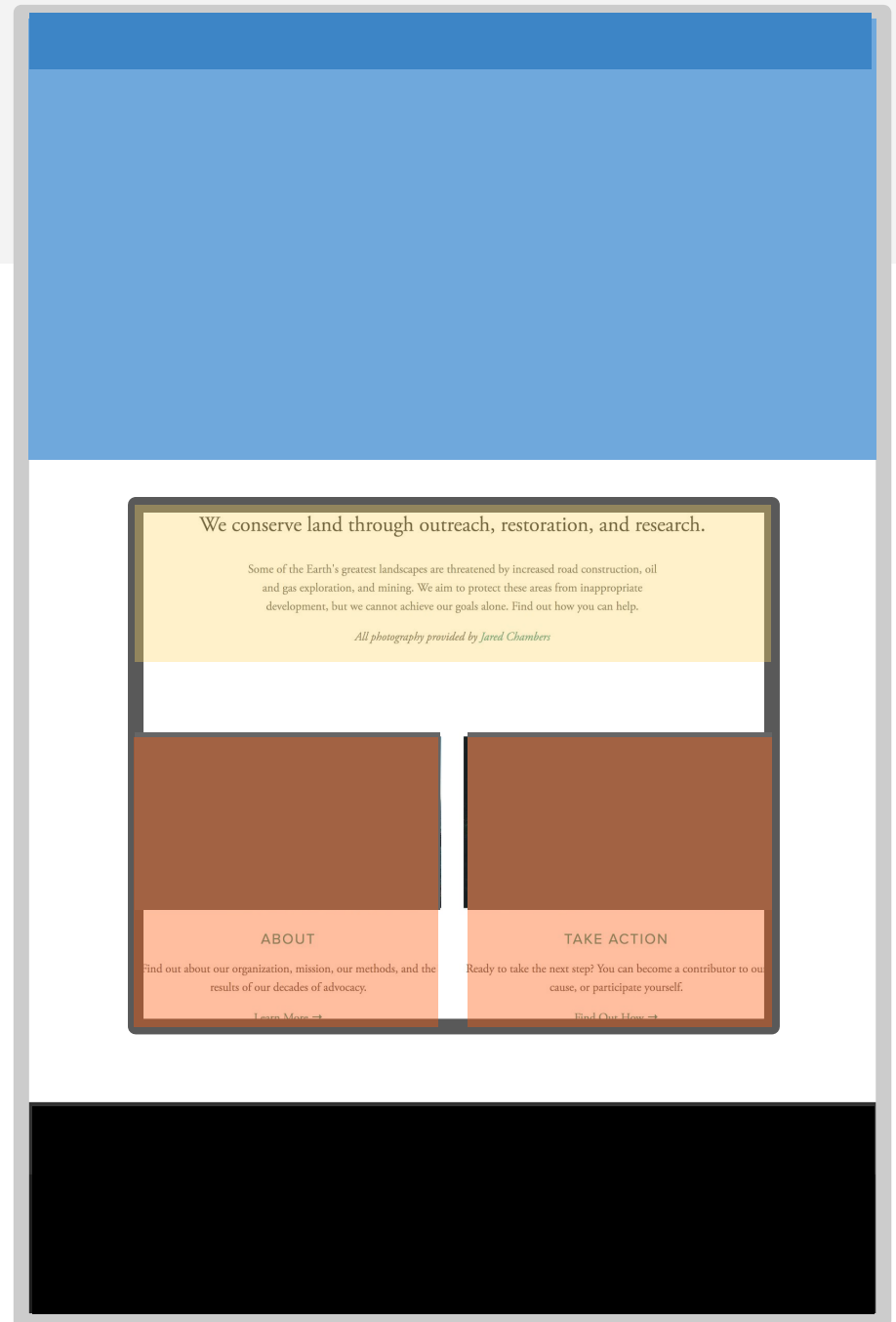
# Basic shape

Begin visualizing the layout in terms of boxes:



# Basic shape

Begin visualizing the layout in terms of boxes:

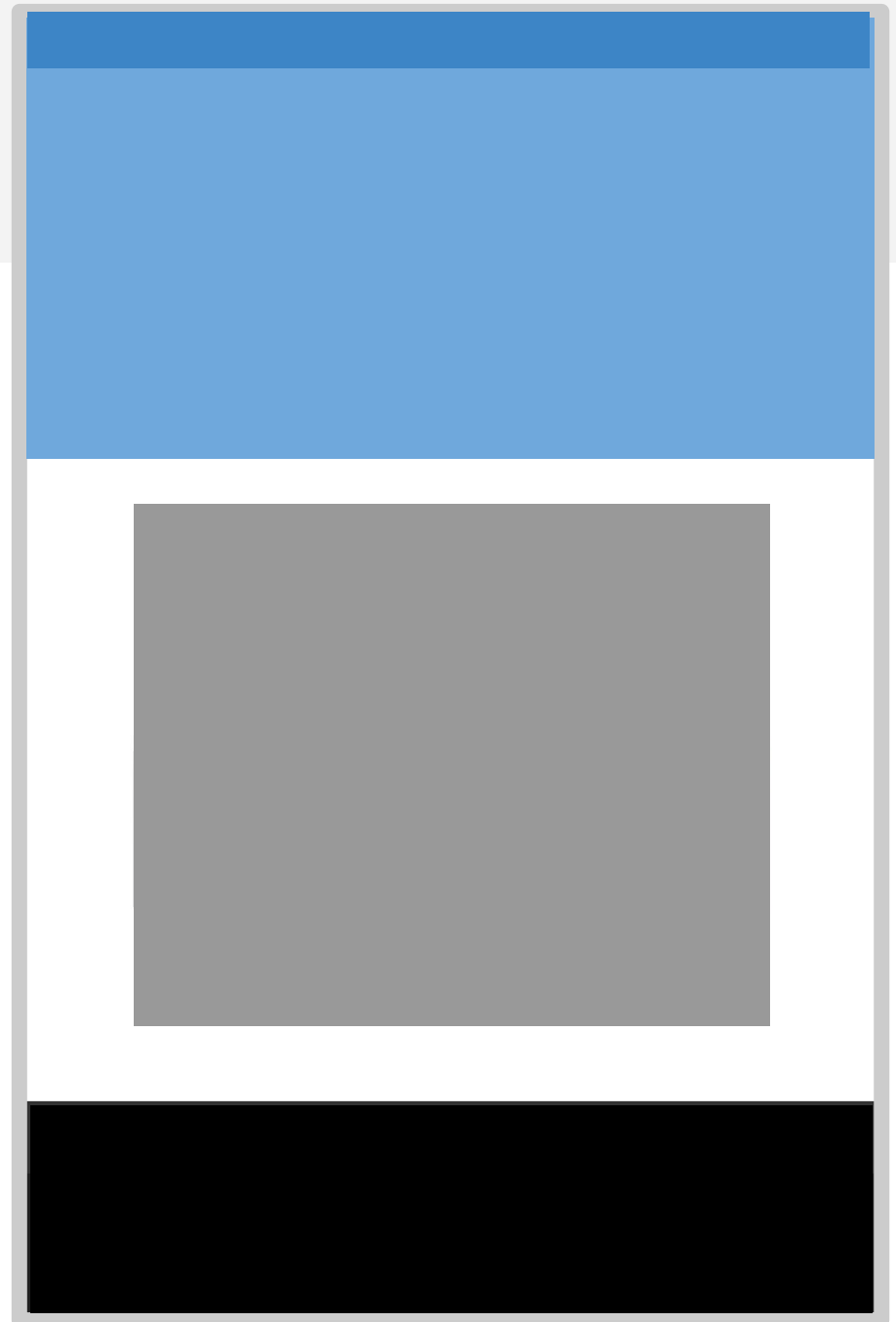


# Basic shape

Begin visualizing the layout in terms of boxes:

Let's first try making this layout!

✦ [Codepen Link](#) ✦



# Content Sectioning elements

Name	Description
<code>&lt;p&gt;</code>	Paragraph ( <a href="#">mdn</a> )
<code>&lt;h1&gt; - &lt;h6&gt;</code>	Section headings ( <a href="#">mdn</a> )
<code>&lt;article&gt;</code>	A document, page, or site ( <a href="#">mdn</a> ) This is usually a root container element after body.
<code>&lt;section&gt;</code>	Generic section of a document ( <a href="#">mdn</a> )
<code>&lt;header&gt;</code>	Introductory section of a document ( <a href="#">mdn</a> )
<code>&lt;footer&gt;</code>	Footer at end of a document or section ( <a href="#">mdn</a> )
<code>&lt;nav&gt;</code>	Navigational section ( <a href="#">mdn</a> )

These elements do not "do" anything; they are basically more descriptive `<div>`s. Makes your HTML more readable. See [MDN](#) for more info.

# Header

## Navbar:

- Height: 75px
- Background: royalblue
- `<nav>`

## Header:

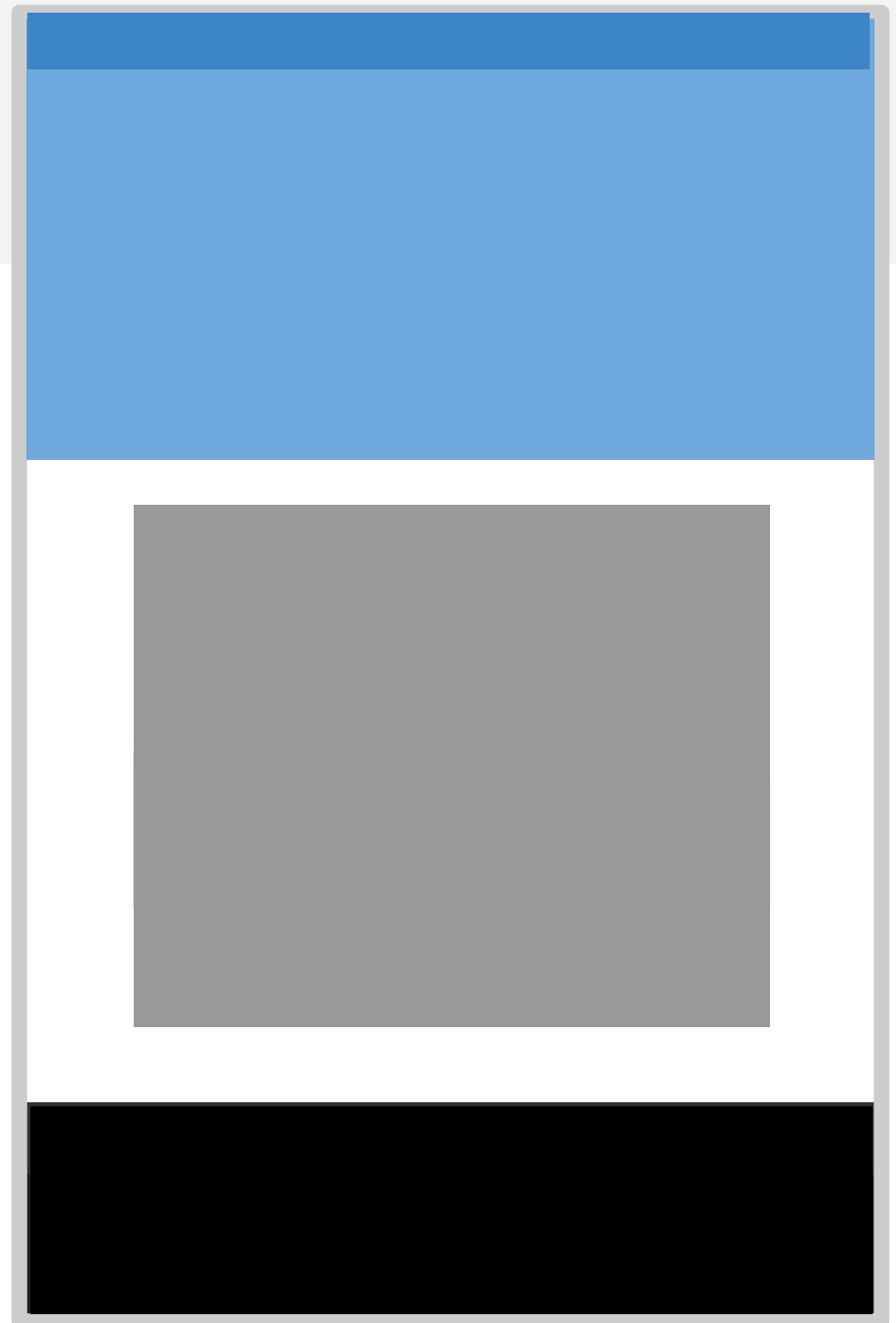
- Height: 400px;
- Background: lightskyblue
- `<header>`



# Main section

## Gray box:

- Surrounding space:  
75px above and  
below; 100px on  
each side
- Height: 500px
- Background: gray
- `<section>`



# Footer

## Footer:

- Height: 100px
- Background: Black
- `<footer>`





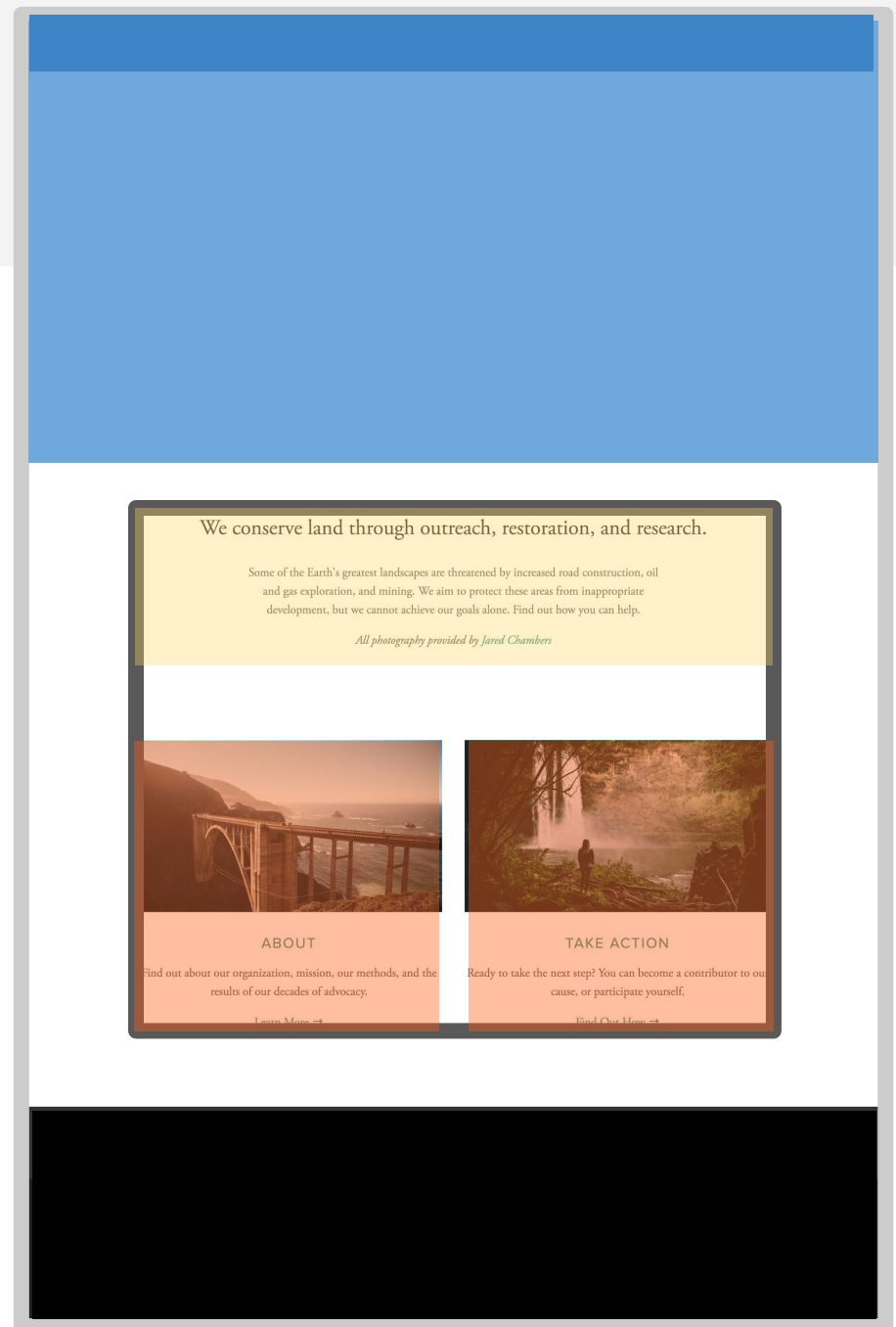
# Main contents

## Yellow paragraph:

- Height: 200px
- Background: khaki
- Space beneath: 75px
- `<p>`

## Orange box:

- Height: 400px;
- Width: 48% of the parent's width, with space in between
- Background: tomato
- `<div>`

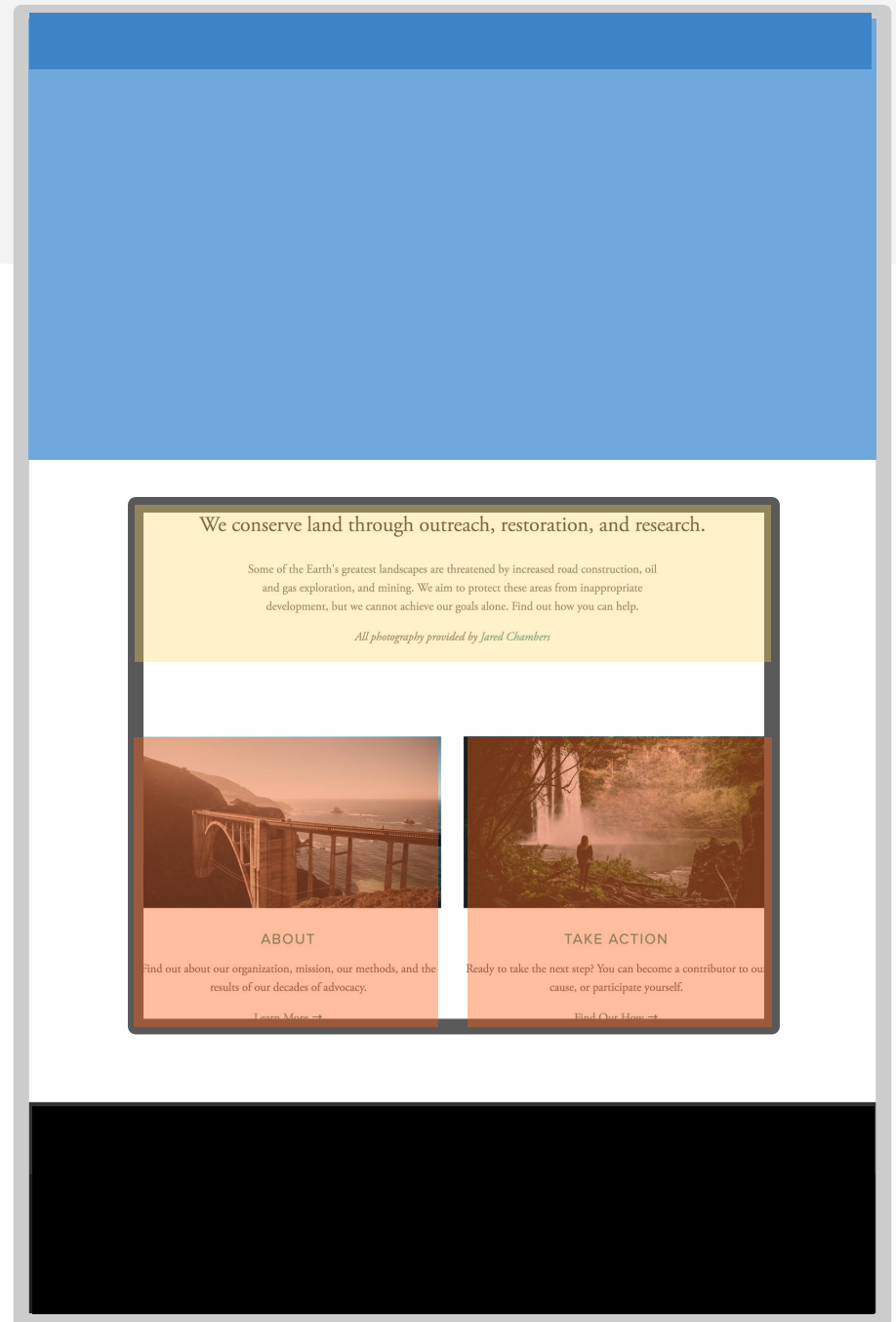


# Main contents

## Orange box:

- Height: 400px;
- Width: 48% of the parent's width, with space in between
- Background: tomato
- `<div>`

**This is where  
we get stuck.**



Next time:  
**Flexbox** to the  
rescue!