

CS193X: Web Programming Fundamentals

Spring 2017

Victoria Kirst
(vrk@stanford.edu)

Today's schedule

Today:

- Wrap up box model
- Debugging with Chrome Inspector
- **Case study:** Squarespace Layout
 - Flex box
 - Misc helpful CSS

Wednesday

- More flexbox; CSS wrap-up

Friday

- Beginning JavaScript

HW1 released

Homework 1 is out today!

- Due Mon Apr 17
- [Details here](#)

Quick review

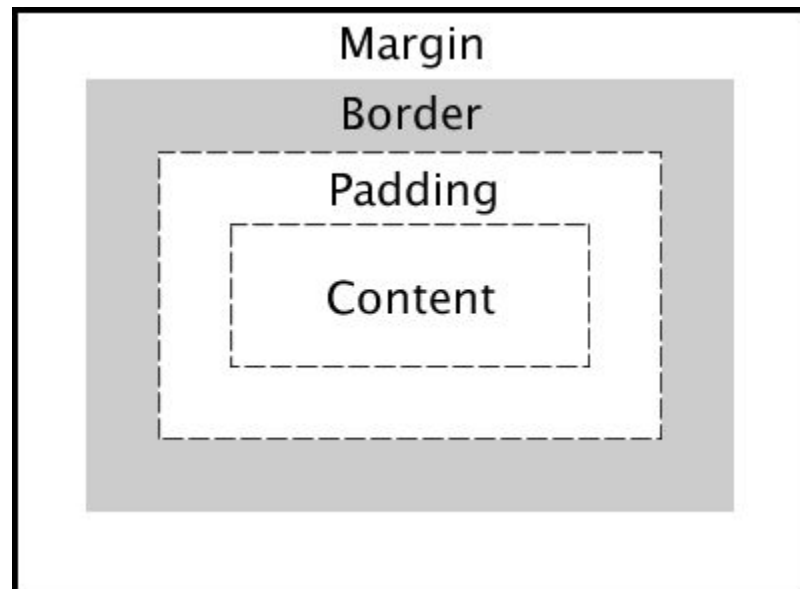
Selector summary

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

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)



<div>s look a little squished

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



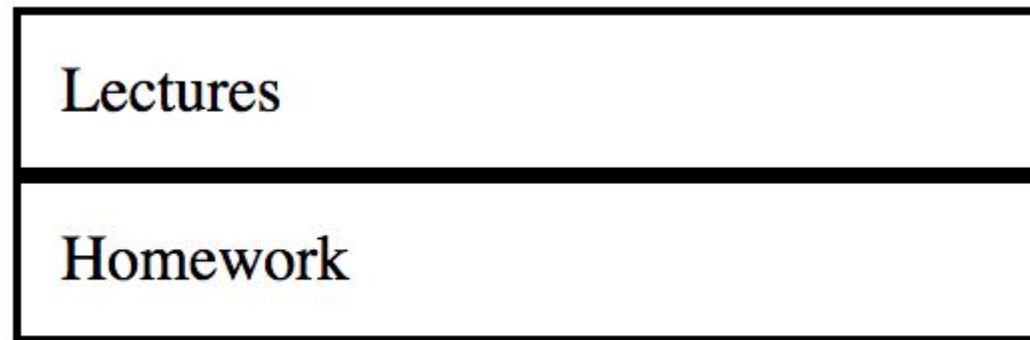
The screenshot shows a code editor with two panels. The left panel, titled 'HTML', contains the following code:

```
<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

Back where we left off!

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

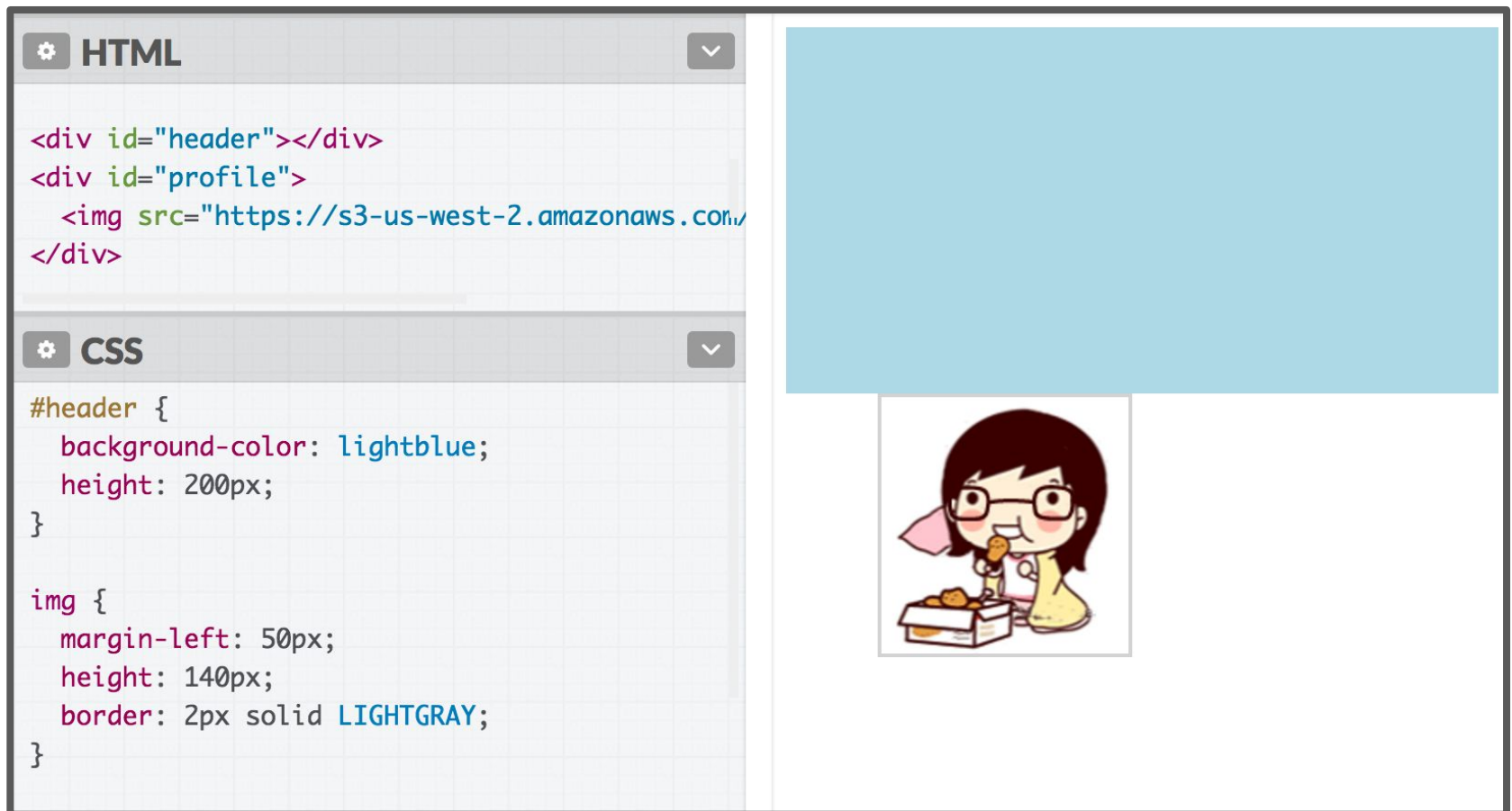
then they collapse into **max**(*Bottom Margin, Top Margin*).

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

Negative margin

Margins can be negative as well.

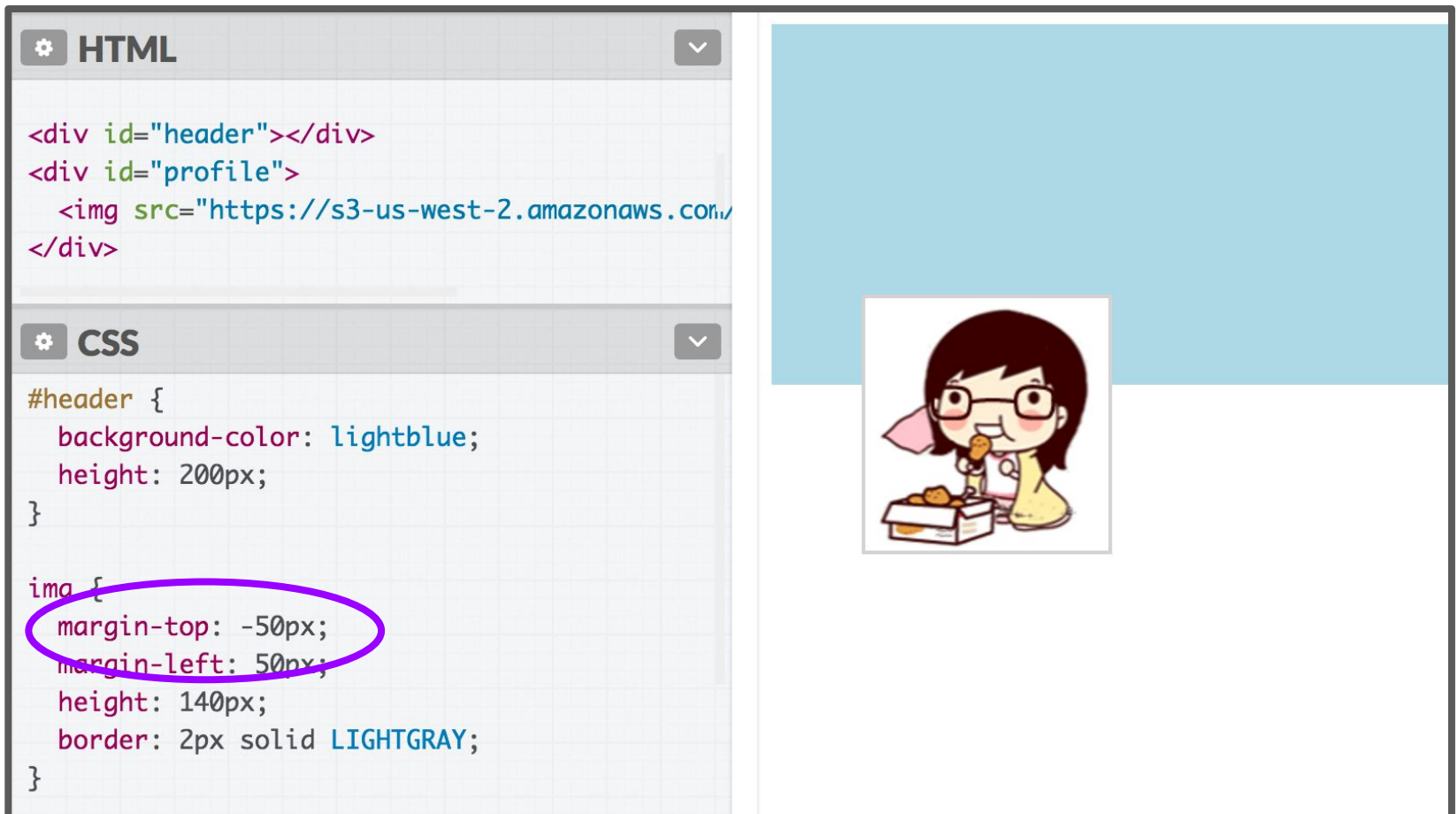
- **No negative margin on image:**



Negative margin

Margins can be negative as well. ([CodePen](#))

- `img { margin-top: -50px; }`



auto margins

If you set `margin-left` and `margin-right` to `auto`, you can center a block-level element ([CodePen](#)):



```
HTML
<html>
  <head>
    <meta charset="utf-8">
    <title>Auto Margins</title>
  </head>
  <body>
    <div>
      This is a box of text.
    </div>
  </body>
</html>

CSS
div {
  margin-left: auto;
  margin-right: auto;
  border: 2px solid black;
  padding: 10px;
  width: 300px;
}
```

This is a box of text.

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;
}

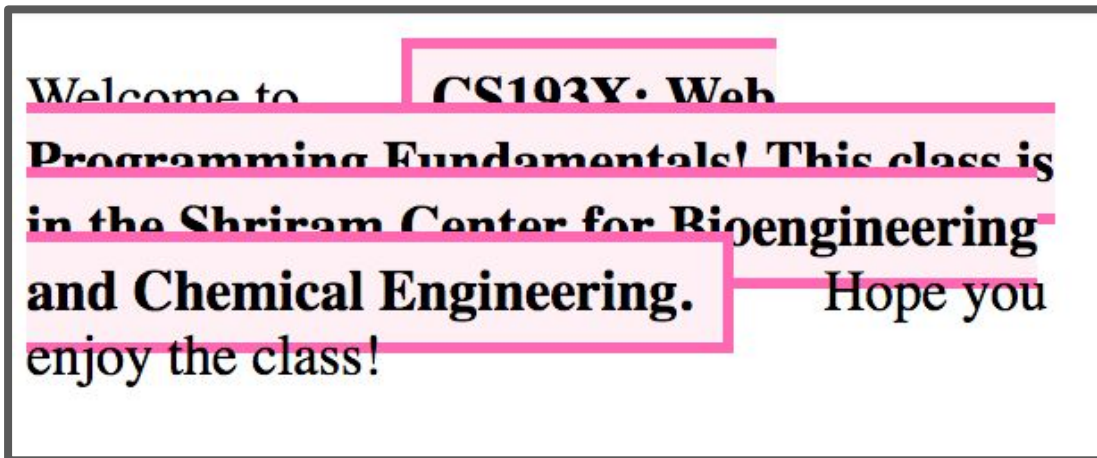
```

```

HTML

<p>
  Welcome to
  <strong>
    CS193X: Web Programming
  </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;
}

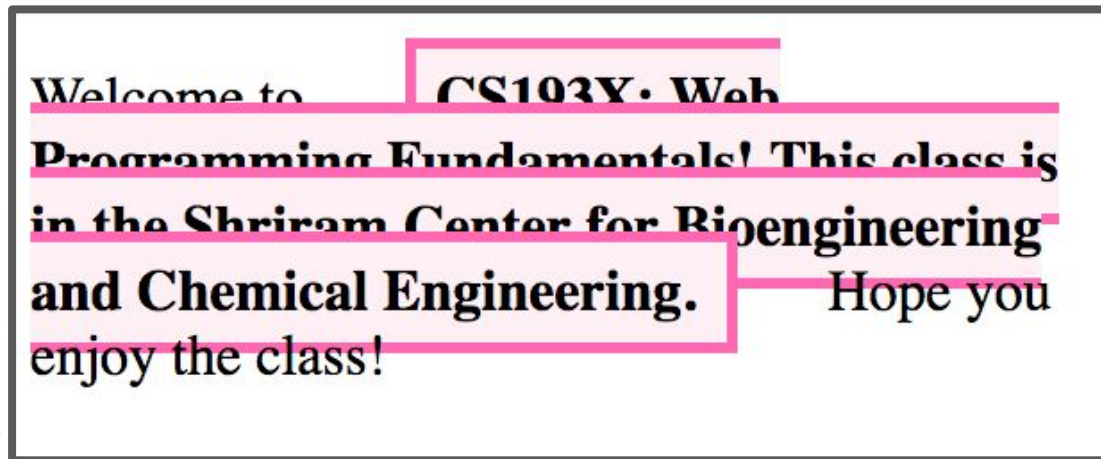
```

```

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](#))

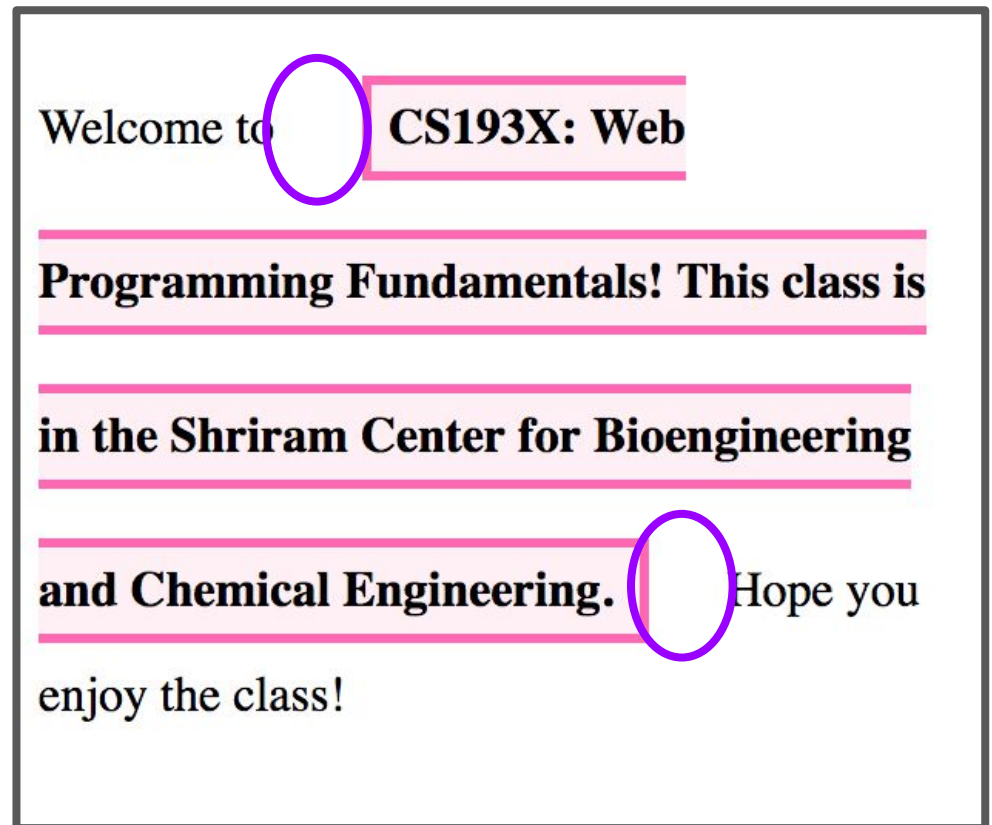
Inline element box model

```

- 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:

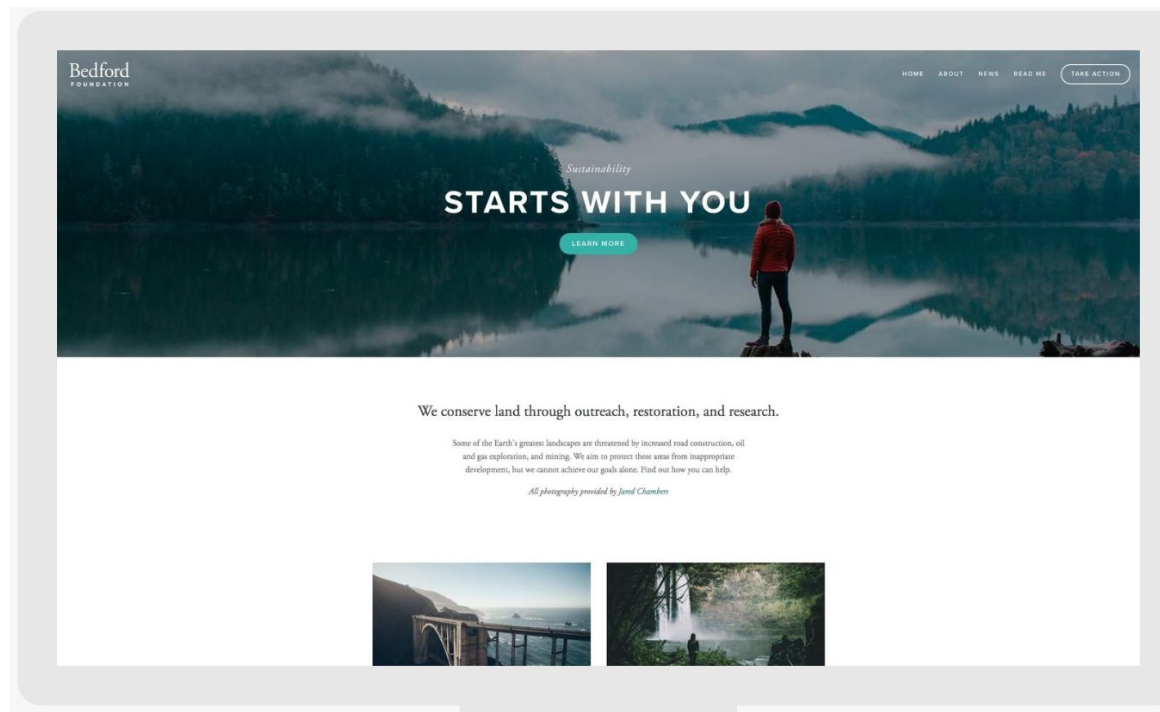
- **block vs inline** and the "natural" layout of the page, depending on the element type
- **classes and ids** and how to specify specific elements and groups of elements
- **div and span** and how to create generic elements
- **The CSS box model** and how every element is shaped like a box, with content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

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

Layout exercise

Squarespace template

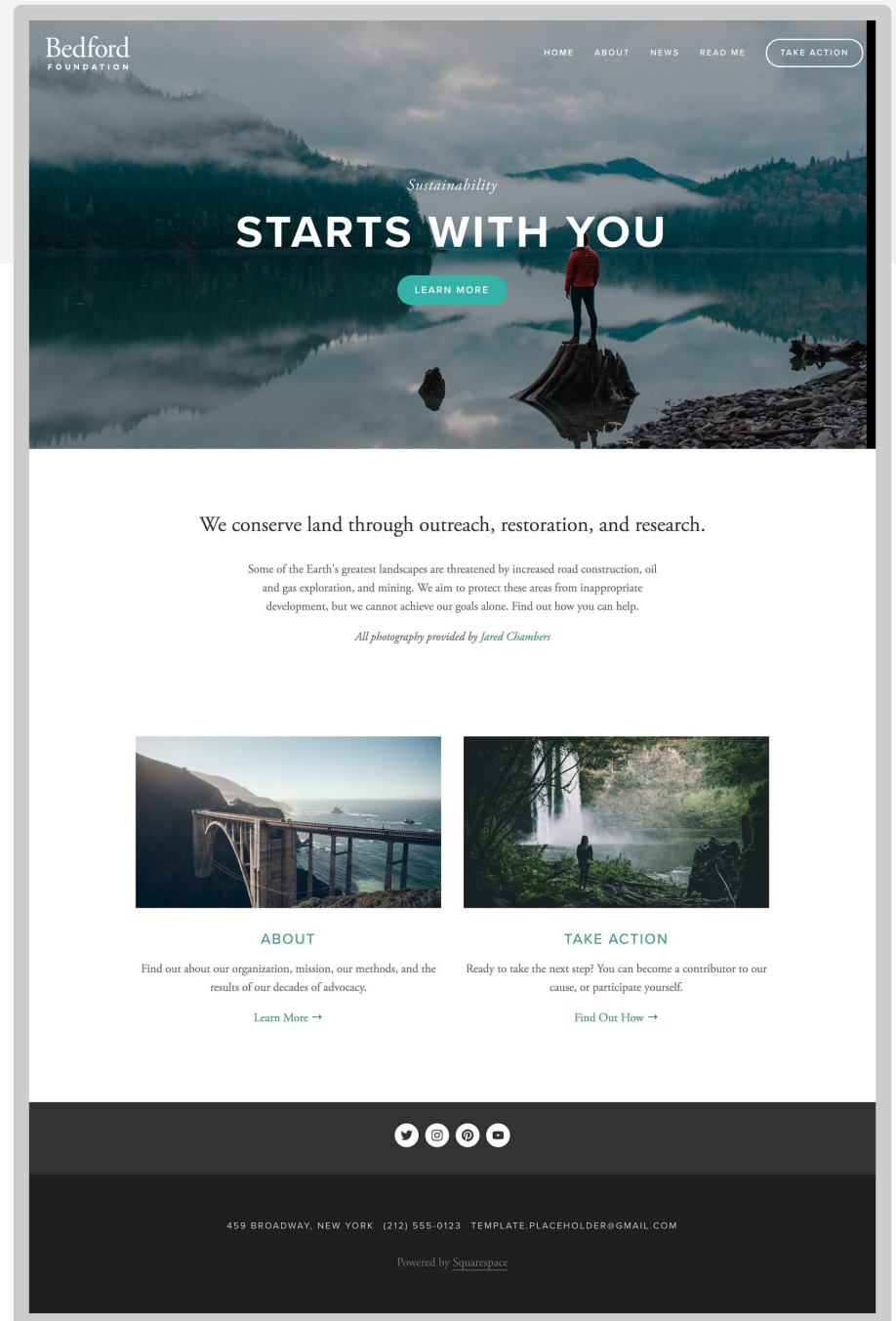
[Squarespace](#)'s most popular template looks like [this](#):



Q: Do we know enough to make something like that?

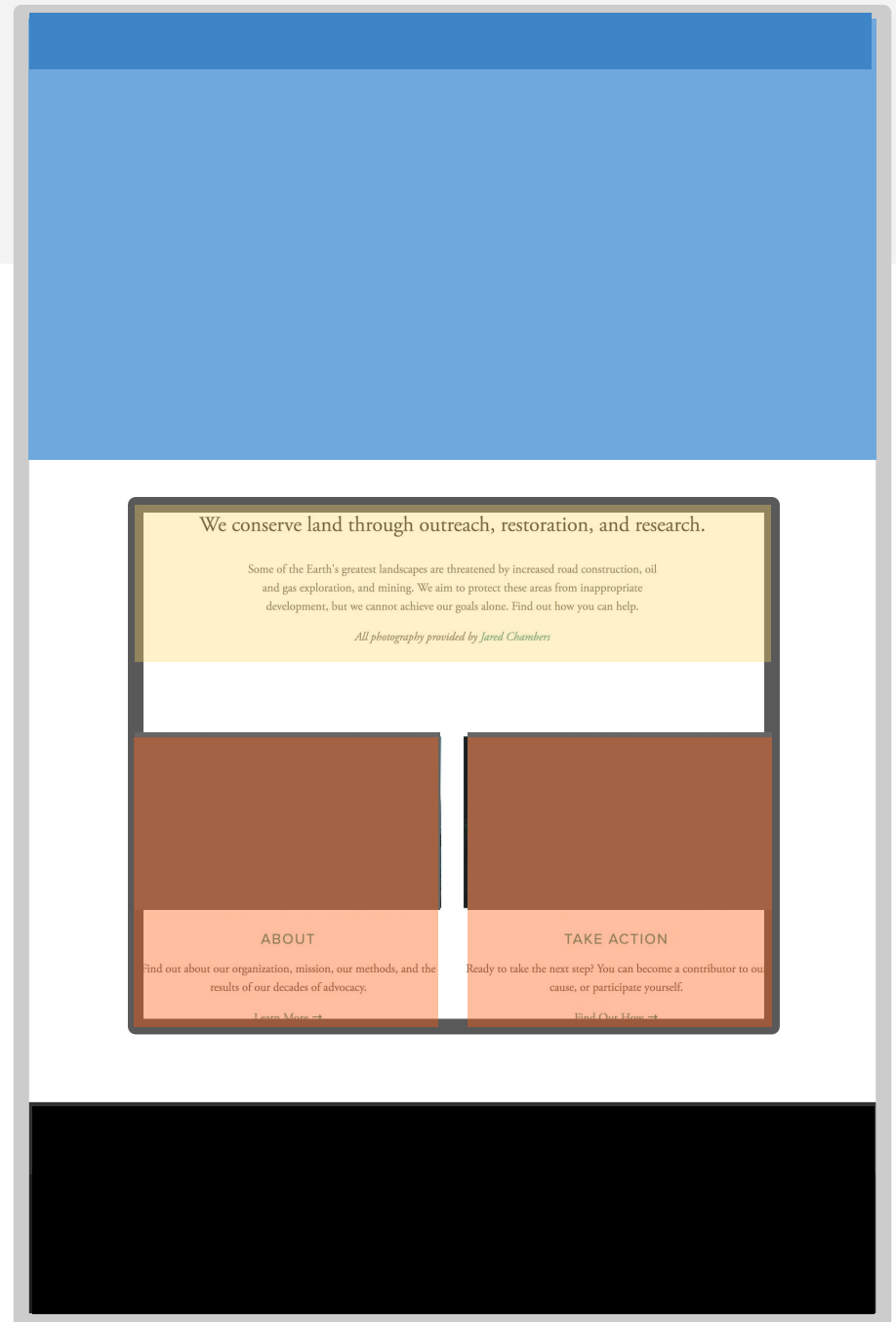
Basic shape

Begin visualizing the layout in terms of boxes:



Basic shape

Begin visualizing the layout in terms of boxes:

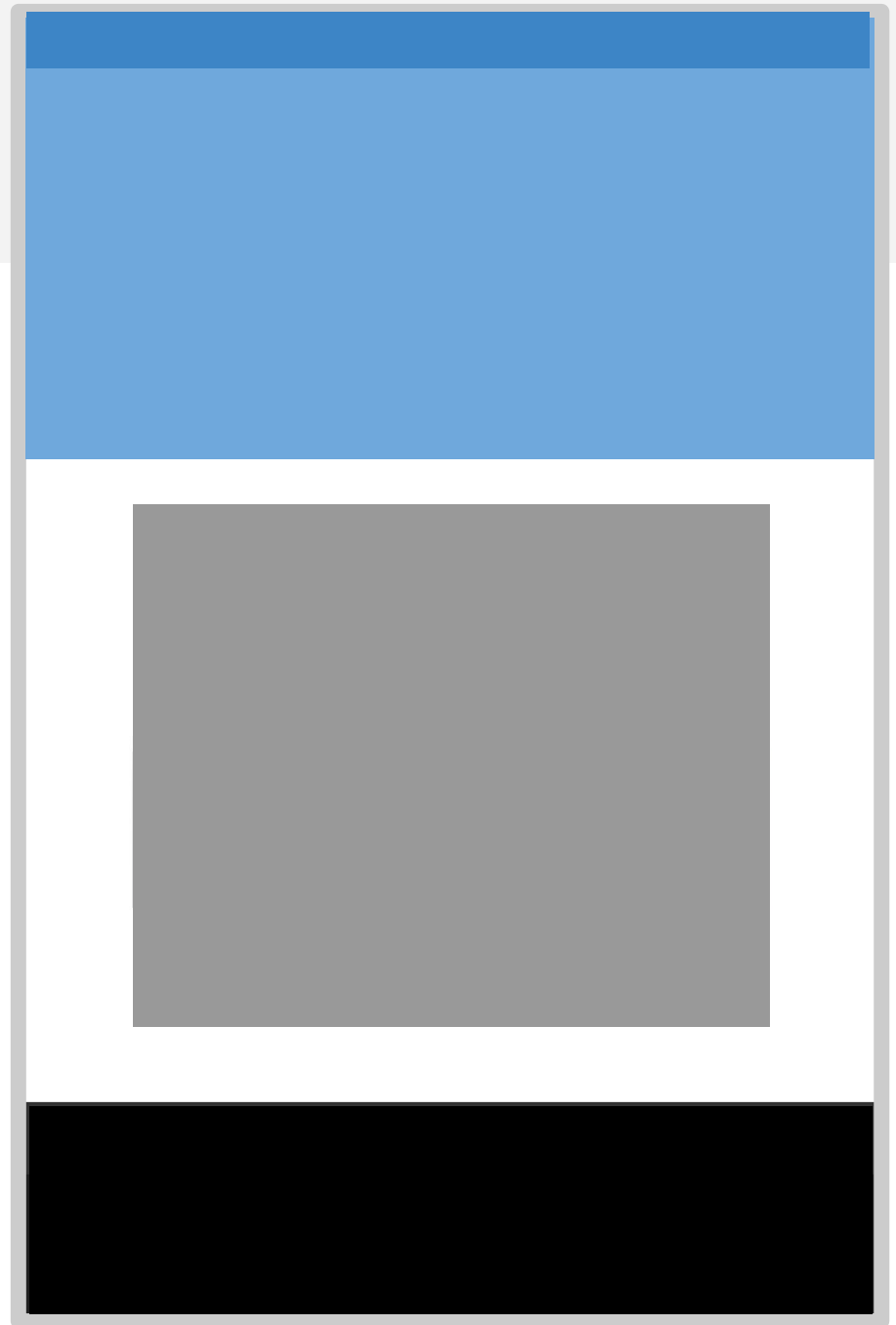


Basic shape

Begin visualizing the layout in terms of boxes:

Let's first try making this layout!

✦ [Codepen Link](#) ✦



Content Sectioning elements

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

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:

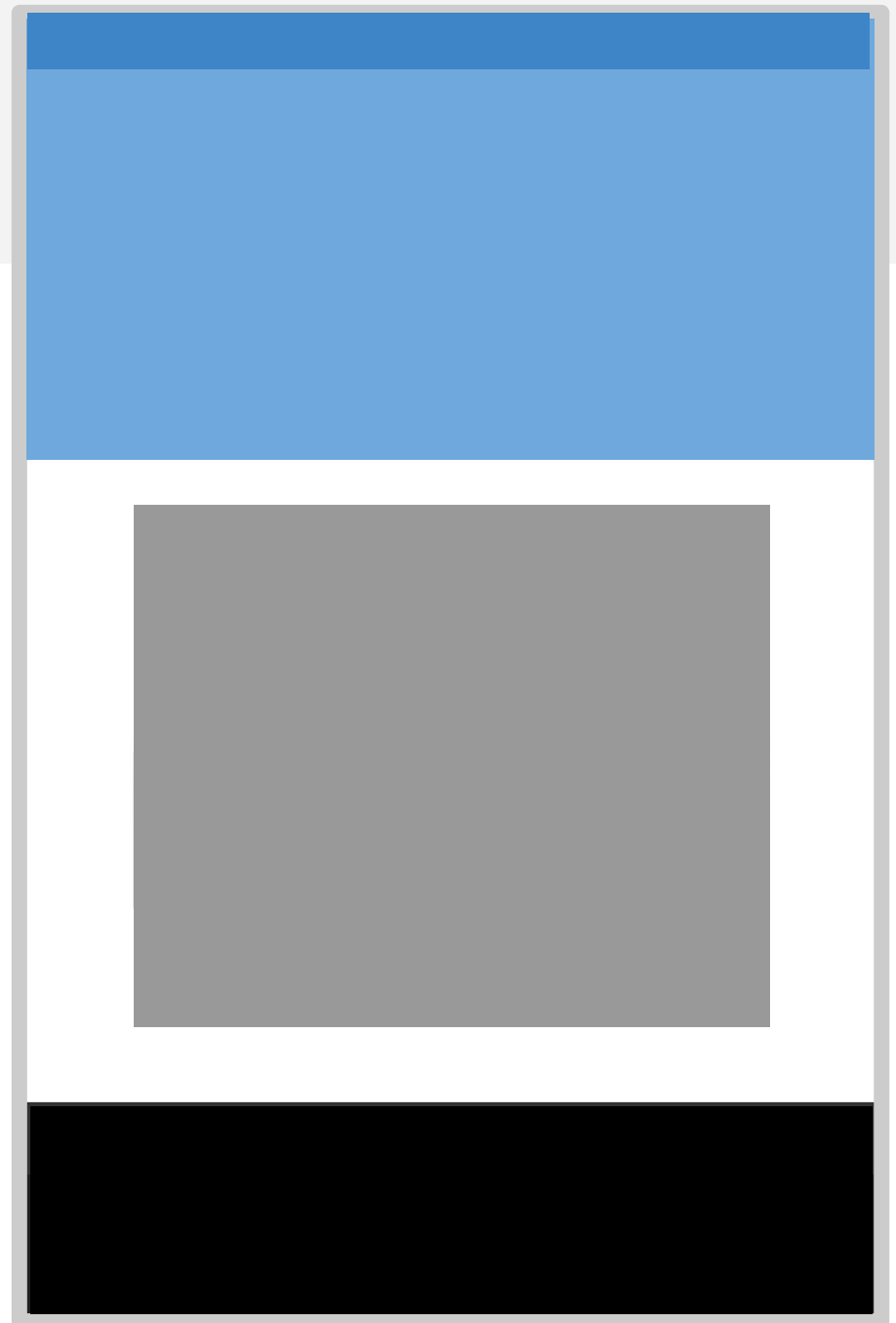
- Height: 400px;
- Background: lightskyblue
- `<header>`



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>`



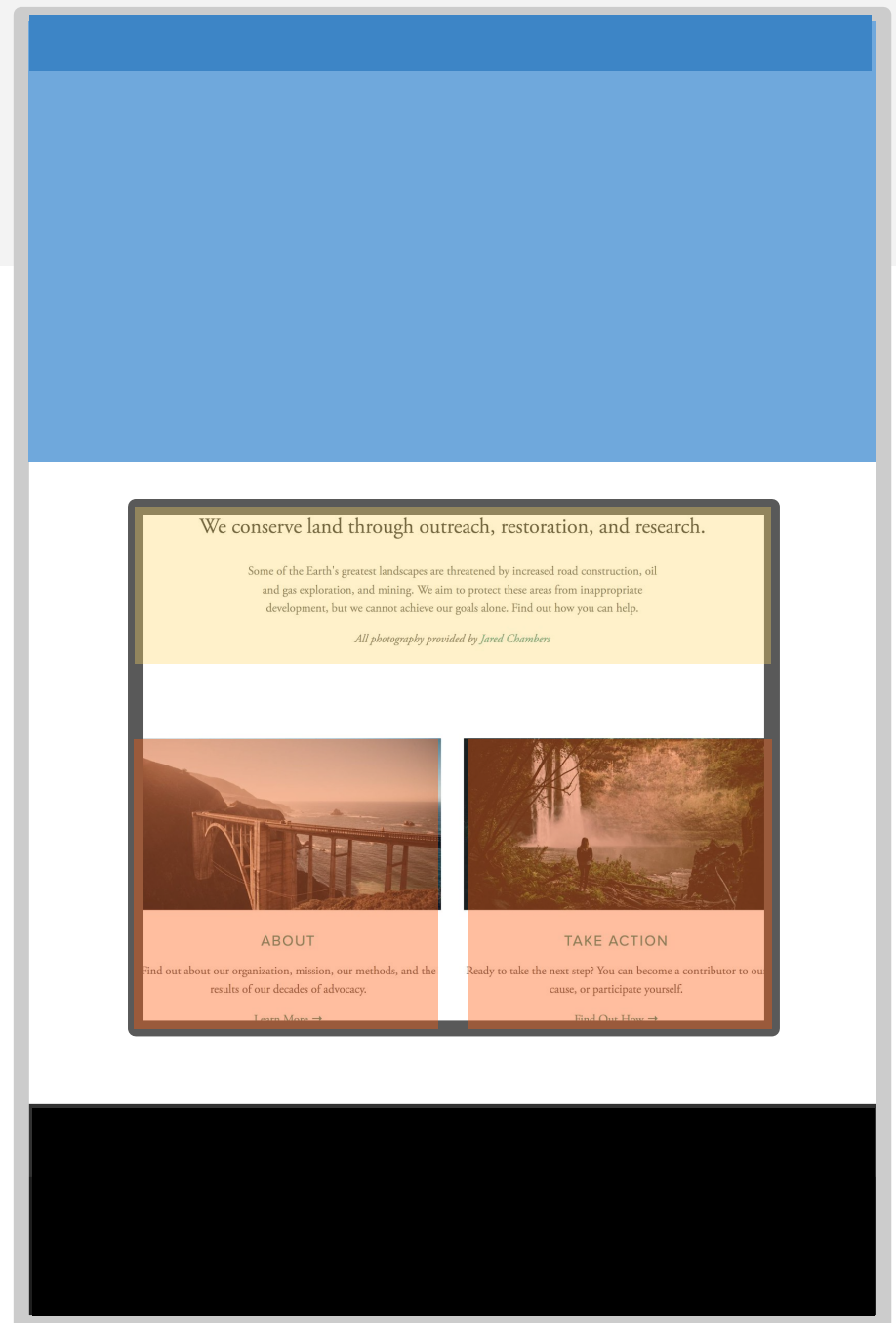
Main contents

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>`

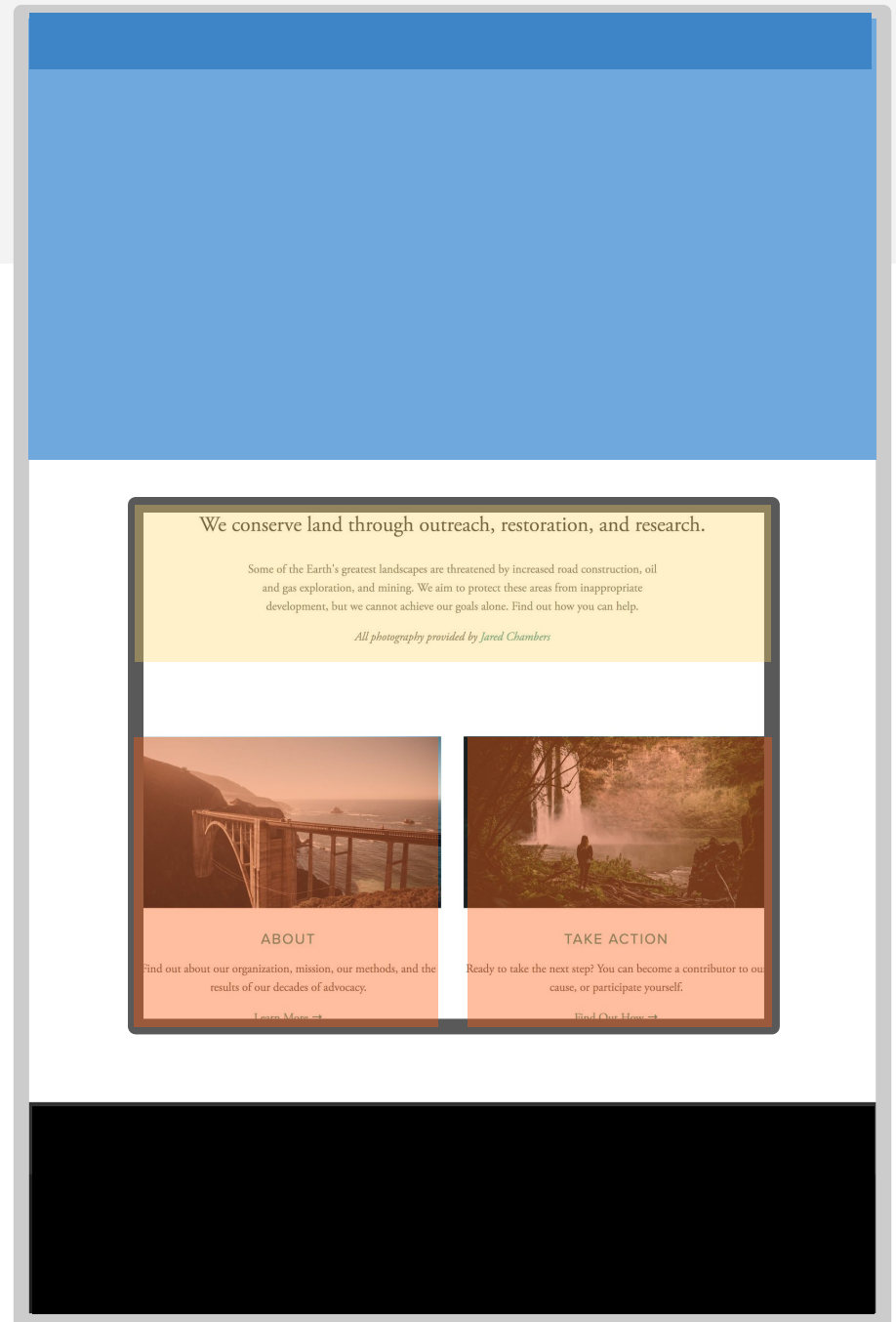


Main contents

Orange box:

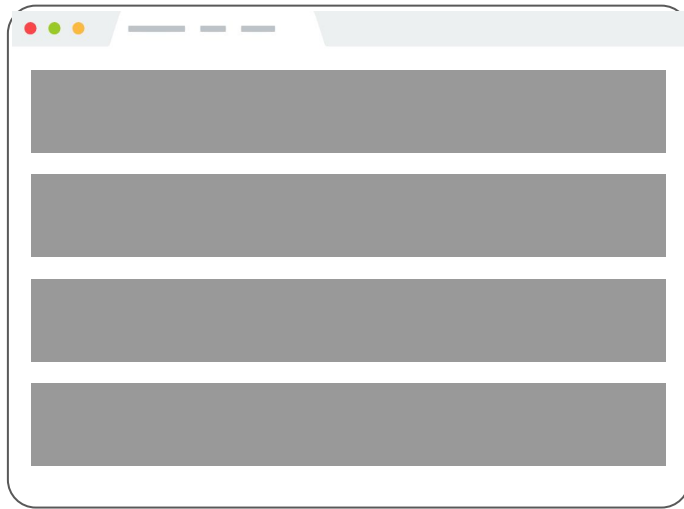
- Height: 400px;
- Width: 48% of the parent's width, with space in between
- Background: tomato
- `<div>`

**This is where
we get stuck.**

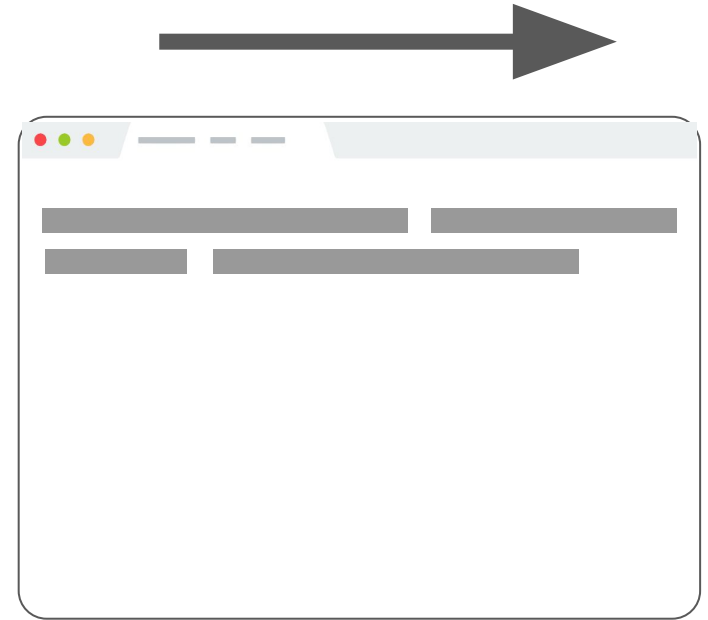


Flexbox

CSS layout so far



Block layout:
Laying out large
sections of a page



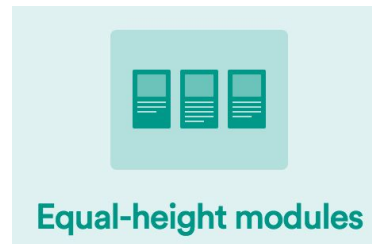
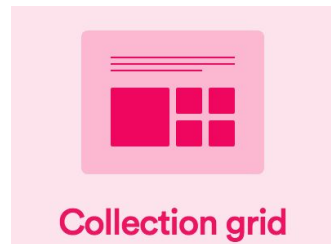
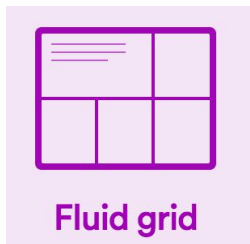
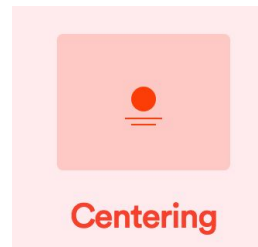
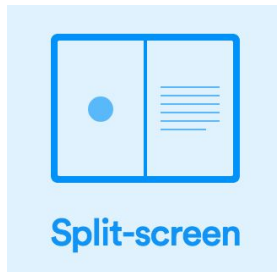
Inline layout:
Laying out text and
other inline content
within a section

Flex layout

To achieve more complicated layouts, we can enable a different kind of CSS layout rendering mode: **Flex layout**.

Flex layout solves all sorts of layout problems.

- Here are some examples of layouts that are easy to create with flex layout (and really difficult otherwise):



**But today we will
only cover the
veeeeeeeery
basics!**


Flex basics

Flex layouts are composed of:

- A **Flex container**, which contains one or more:
 - **Flex item**(s)

You can then apply CSS properties on the **flex container** to dictate how the flex items are displayed.

id=flex-container



A diagram illustrating a flex container and its item. A large blue-outlined rectangle represents the flex container. Inside the top-left corner of this container is a smaller, rounded purple rectangle representing a flex item. The text 'class= flex-item' is written inside the purple rectangle.

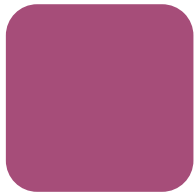
```
class=
flex-
item
```


Flex basics

To make an element a flex container, change `display`:

- Block container: `display: flex;` or
- Inline container: `display: inline-flex;`

Follow along in [Codepen](#)



HTML

```
<html>
  <head>
    <meta charset="utf-8">
    <title>Flexbox example</title>
  </head>
  <body>

    <div id="flex-container">
      <div class="flex-item"></div>
    </div>

  </body>
</html>
```

CSS

```
#flex-container {
  display: flex;
  border: 2px solid black;
  padding: 10px;
  height: 150px;
}

.flex-item {
  border-radius: 10px;
  background-color: purple;
  height: 50px;
  width: 50px;
}
```

JS



HTML

```
<html>
  <head>
    <meta charset="utf-8">
    <title>Flexbox example</title>
  </head>
  <body>

    <div id="flex-container">
      <div class="flex-item"></div>
    </div>

  </body>
</html>
```

CSS

```
#flex-container {
  display: flex;
  border: 2px solid black;
  padding: 10px;
  height: 150px;
}

.flex-item {
  border-radius: 10px;
  background-color: purple;
  height: 50px;
}
```

JS



(So far, this looks exactly the same as `display: block`)

Flex basics: justify-content

You can control where the item is horizontally* in the box by setting `justify-content` on the flex container:

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

*when flex direction is row. We'll get to what "flex direction" means soon.



Flex basics: justify-content

You can control where the item is horizontally* in the box by setting `justify-content` on the flex container:

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

*when flex direction is row. We'll get to what "flex direction" means soon.

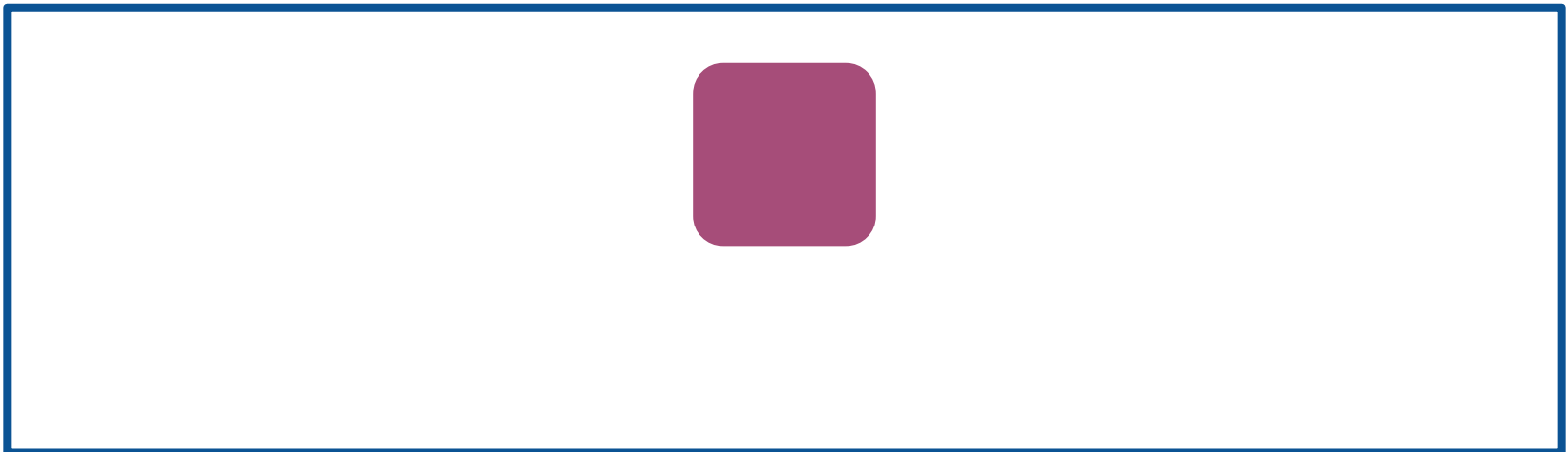


Flex basics: justify-content

You can control where the item is horizontally* in the box by setting `justify-content` on the flex container:

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

*when flex direction is row. We'll get to what "flex direction" means soon.



Flex basics: align-items

You can control where the item is vertically* in the box by setting `align-items` on the flex container:

```
#flex-container {  
  display: flex;  
  align-items: flex-start;  
}
```

*when flex direction is row. We'll get to what "flex direction" means soon.



Flex basics: align-items

You can control where the item is vertically* in the box by setting `align-items` on the flex container:

```
#flex-container {  
  display: flex;  
  align-items: flex-end;  
}
```

*when flex direction is row. We'll get to what "flex direction" means soon.



Flex basics: align-items

You can control where the item is vertically* in the box by setting `align-items` on the flex container:

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

*when flex direction is row. We'll get to what "flex direction" means soon.



Multiple items

Same rules apply with multiple flex items:

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



Multiple items

Same rules apply with multiple flex items:

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



Multiple items

Same rules apply with multiple flex items:

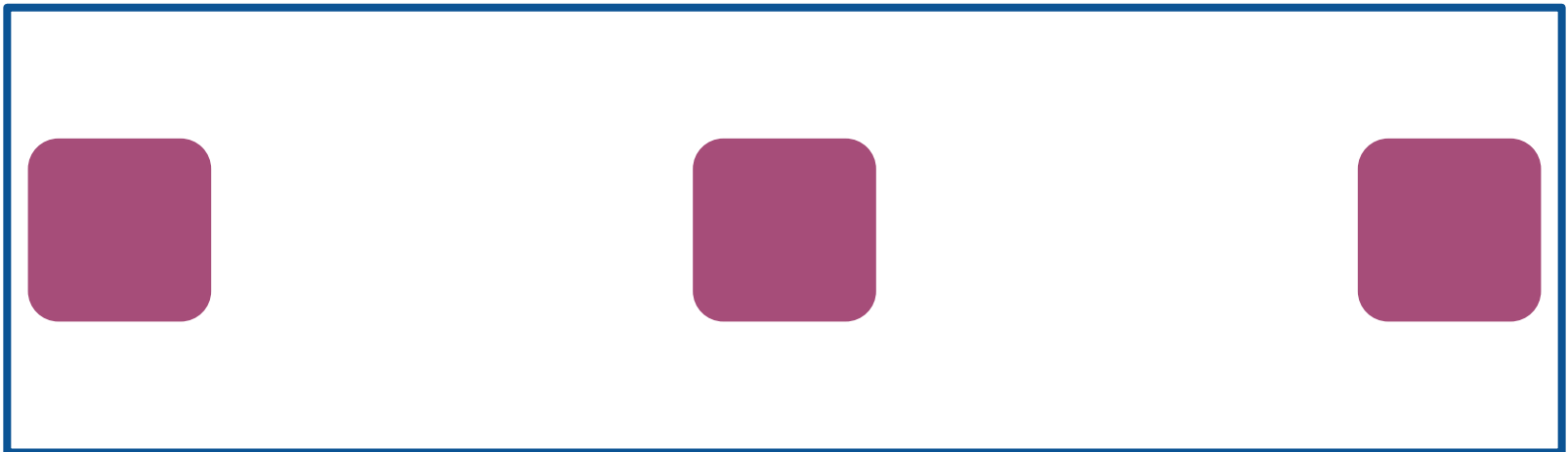
```
#flex-container {  
  display: flex;  
  justify-content: center;  
  align-items: center;  
}
```



Multiple items

And there is also **space-between** and **space-around**:

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



Multiple items

And there is also **space-between** and **space-around**:

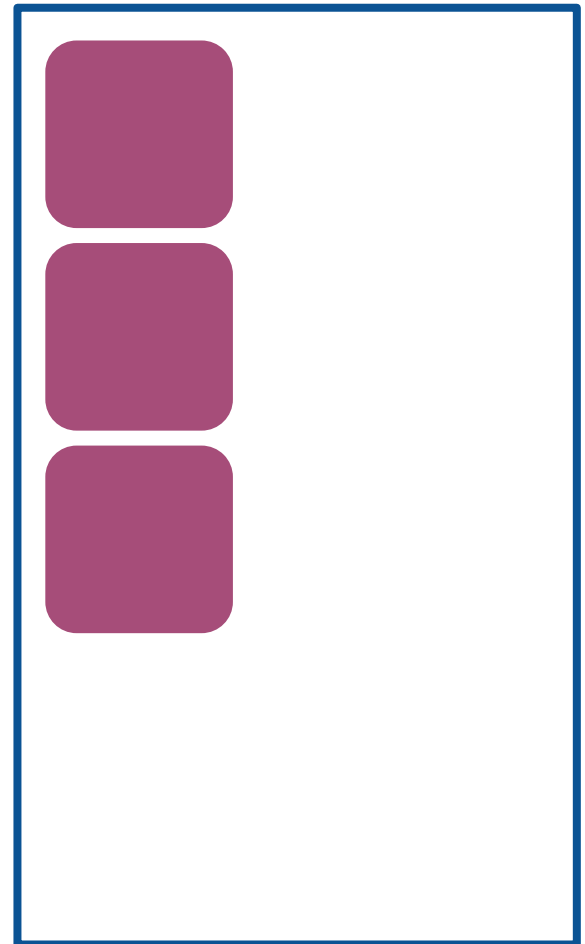
```
#flex-container {  
  display: flex;  
  justify-content: space-around;  
  align-items: center;  
}
```



flex-direction

And you can also lay out columns instead of rows:

```
#flex-container {  
  display: flex;  
  flex-direction: column;  
}
```

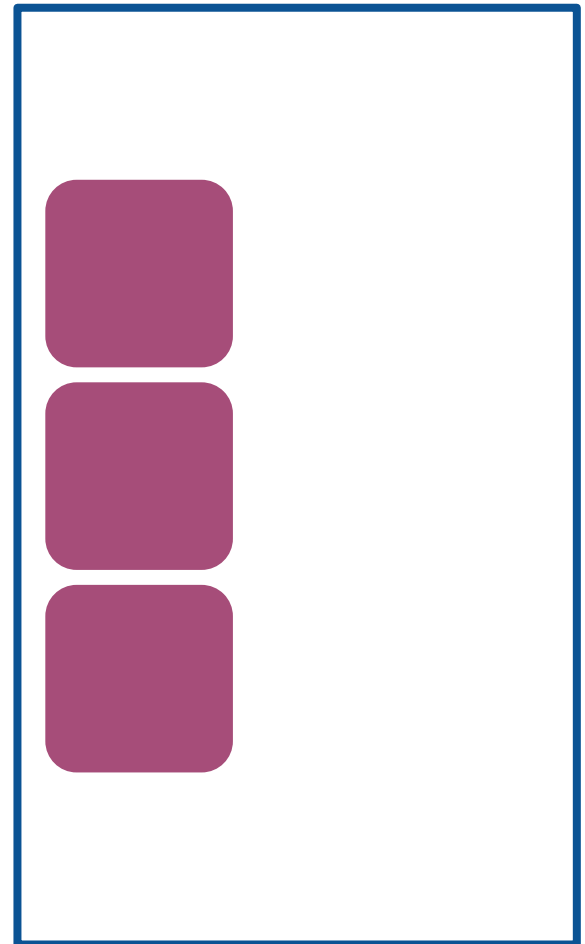


flex-direction

And you can also lay out columns instead of rows:

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

Now **justify-content** controls where the column is vertically in the box

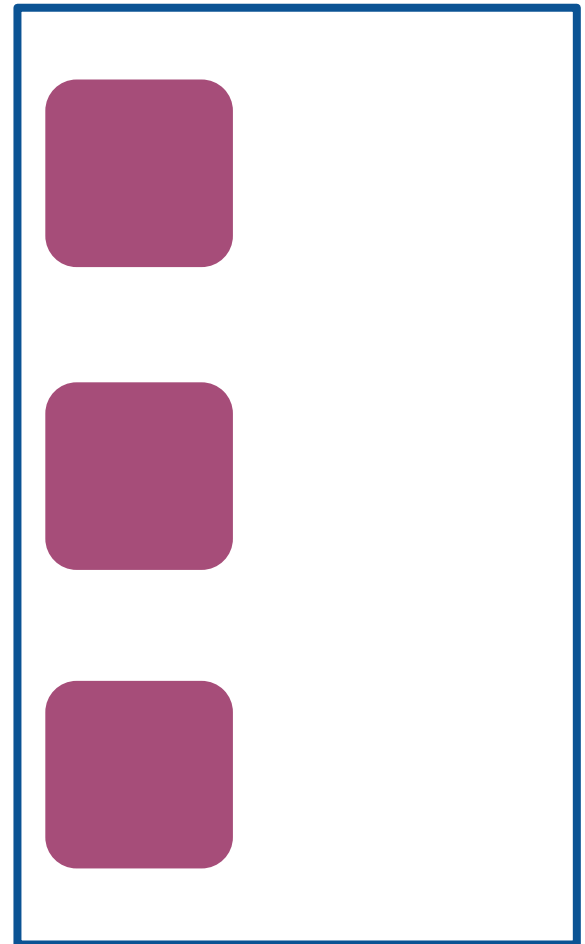


flex-direction

And you can also lay out columns instead of rows:

```
#flex-container {  
  display: flex;  
  flex-direction: column;  
  justify-content: space-around;  
}
```

Now **justify-content** controls where the column is vertically in the box

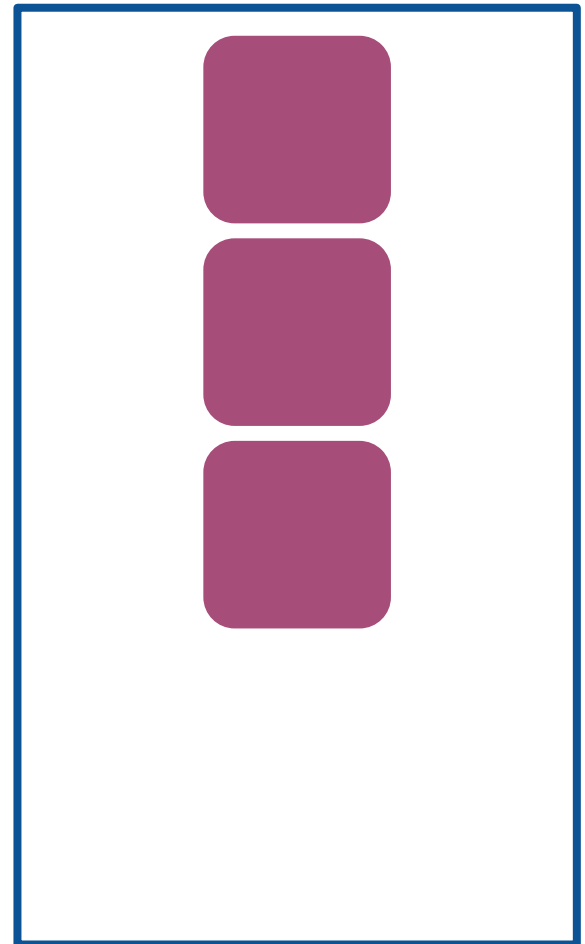


flex-direction

And you can also lay out columns instead of rows:

```
#flex-container {  
  display: flex;  
  flex-direction: column;  
  align-items: center;  
}
```

Now **align-items** controls where the column is horizontally in the box

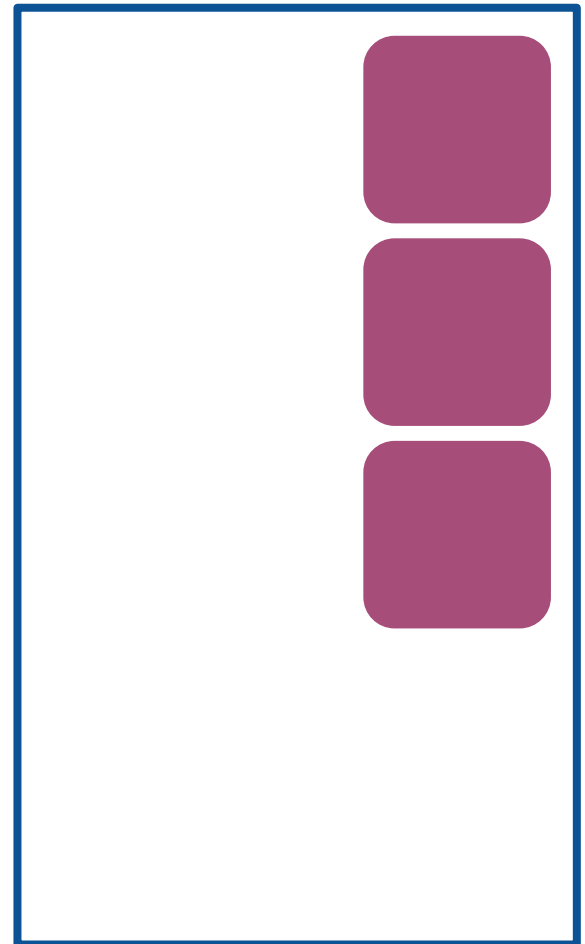


flex-direction

And you can also lay out columns instead of rows:

```
#flex-container {  
  display: flex;  
  flex-direction: column;  
  align-items: flex-end;  
}
```

Now **align-items** controls where the column is horizontally in the box



Before we move on...

What happens if the flex item is an inline element?

HTML

```
<html>
  <head>
    <meta charset="utf-8">
    <title>Flexbox example</title>
  </head>
  <body>

    <div id="flex-container">
      <span class="flex-item"></span>
      <span class="flex-item"></span>
      <span class="flex-item"></span>
    </div>

  </body>
```

CSS

```
#flex-container {
  display: flex;
  border: 2px solid black;
  height: 150px;
}

.flex-item {
  border-radius: 10px;
  background-color: purple;
  height: 50px;
  width: 50px;
  margin: 5px;
}
```

JS

???

HTML

```
<html>
  <head>
    <meta charset="utf-8">
    <title>Flexbox example</title>
  </head>
  <body>

    <div id="flex-container">
      <span class="flex-item"></span>
      <span class="flex-item"></span>
      <span class="flex-item"></span>
    </div>

  </body>
```

CSS

```
#flex-container {
  display: flex;
  border: 2px solid black;
  height: 150px;
}

.flex-item {
  border-radius: 10px;
  background-color: purple;
  height: 50px;
  width: 50px;
  margin: 5px;
}
```



Recall: block layouts

If #flex-container was **not** display: flex:



Then the span flex-items would not show up because span elements are inline, which don't have a height and width

Flex layouts



Why does this change when `display: flex`?

Why do inline elements suddenly seem to have height and width?

Flex: A different rendering mode

- When you set a container to `display: flex`, the **direct children in that container are flex items** and follow a new set of rules.
- **Flex items are not block or inline**; they have different rules for their height, width, and layout.
 - The *contents* of a flex item follow the usual block/inline rules, relative to the flex item's boundary.
- The **height** and **width** of flex items are... complicated.

Flex item sizing

Flex basis

Flex items have an initial width*, which, by default is either:

- The content width, or
- The explicitly set **width** property of the element, or
- The explicitly set **flex-basis** property of the element

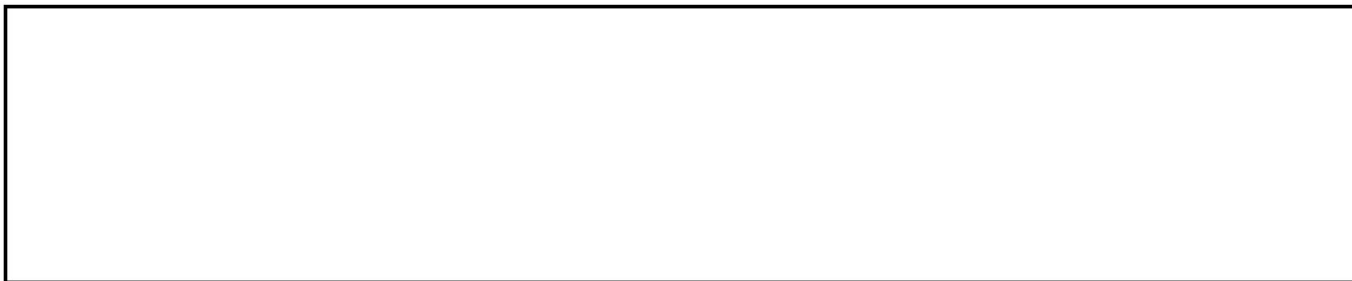
This initial width* of the flex item is called the **flex basis**.

*width in the case of rows; height in
the case of columns

Flex basis

If we unset the height and width, our flex items disappears, because the **flex basis** is now the content size, which is empty:

HTML	CSS
<pre><title>Flexbox example</title> </head> <body> <div id="flex-container"> <div class="flex-item"></div> </div> </body> </html></pre>	<pre>#flex-container { display: flex; border: 2px solid black; height: 150px; } .flex-item { border-radius: 10px; background-color: purple; margin: 5px; }</pre>



flex-shrink

The width* of the flex item can automatically shrink **smaller than the flex basis** via the **flex-shrink** property:

flex-shrink:

- If set to 1, the flex item shrinks itself as small as it can in the space available.
- If set to 0, the flex item does not shrink.

Flex items have **flex-shrink: 1 by default.**

*width in the case of rows; height in the case of columns

```
#flex-container {  
  display: flex;  
  align-items: flex-start;  
  border: 2px solid black;  
  height: 150px;  
}
```

```
.flex-item {  
  width: 500px;  
  height: 100px;  
  
  border-radius: 10px;  
  background-color: purple;  
  margin: 5px;  
}
```



The flex items' widths all shrink to fit within the container.

```
#flex-container {  
  display: flex;  
  align-items: flex-start;  
  border: 2px solid black;  
  height: 150px;  
}
```

```
.flex-item {  
  width: 500px;  
  height: 100px;  
  flex-shrink: 0;  
  
  border-radius: 10px;  
  background-color: purple;  
  margin: 5px;  
}
```

Setting `flex-shrink: 0;` undoes the shrinking behavior, and the flex items do not shrink in any circumstance:



flex-grow

The width* of the flex item can automatically **grow larger than the flex basis** via the **flex-grow** property:

flex-grow:

- If set to 1, the flex item grows itself as large as it can in the space remaining.
- If set to 0, the flex-item does not grow.

Flex items have **flex-grow: 0 by default.**

*width in the case of rows; height in the case of columns

flex-grow example

Let's unset the height and width of our flex items again:

HTML

```
<title>Flexbox example</title>
</head>
<body>

  <div id="flex-container">
    <span class="flex-item"></span>
    <div class="flex-item"></div>
    <span class="flex-item"></span>
  </div>

</body>
</html>
```

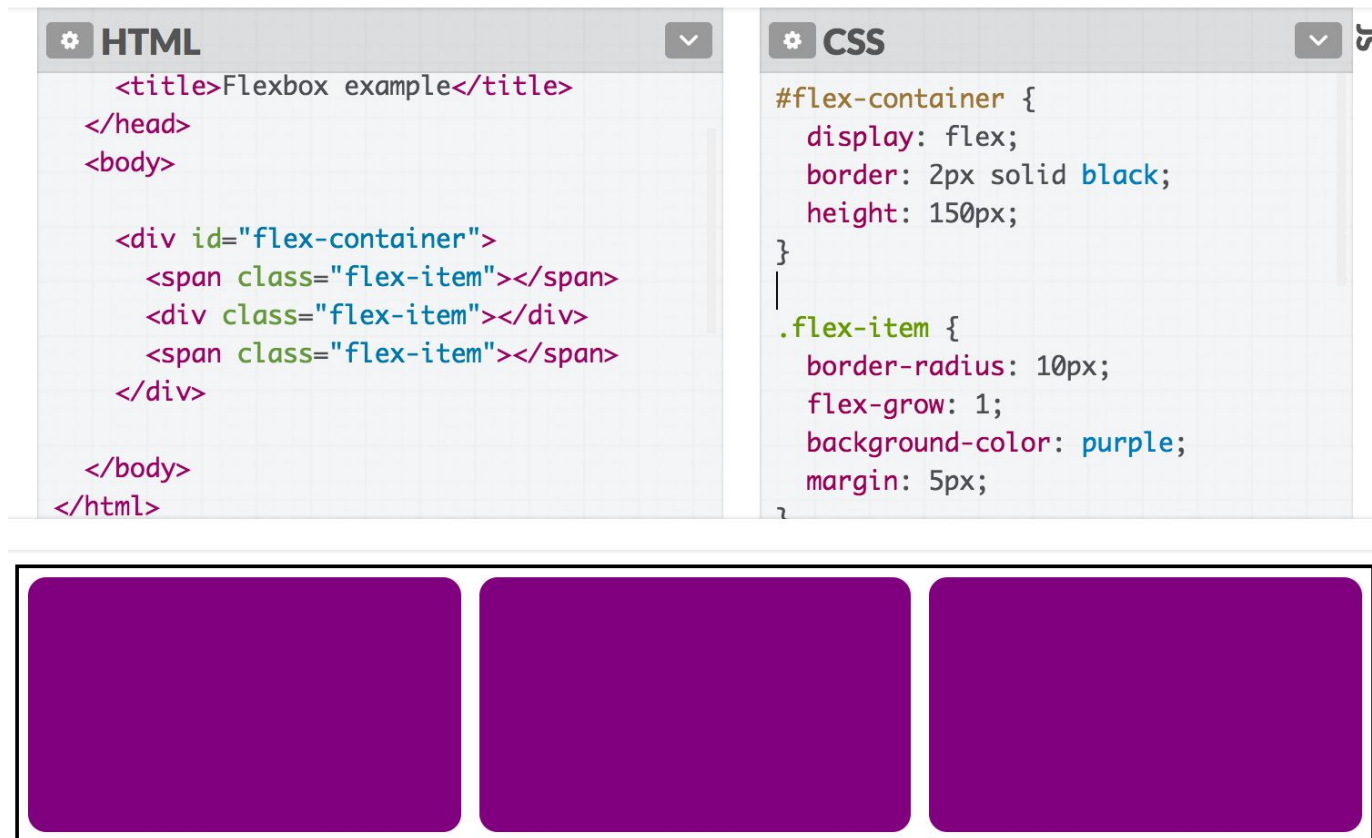
CSS

```
#flex-container {
  display: flex;
  border: 2px solid black;
  height: 150px;
}

.flex-item {
  border-radius: 10px;
  background-color: purple;
  margin: 5px;
}
```

flex-grow example

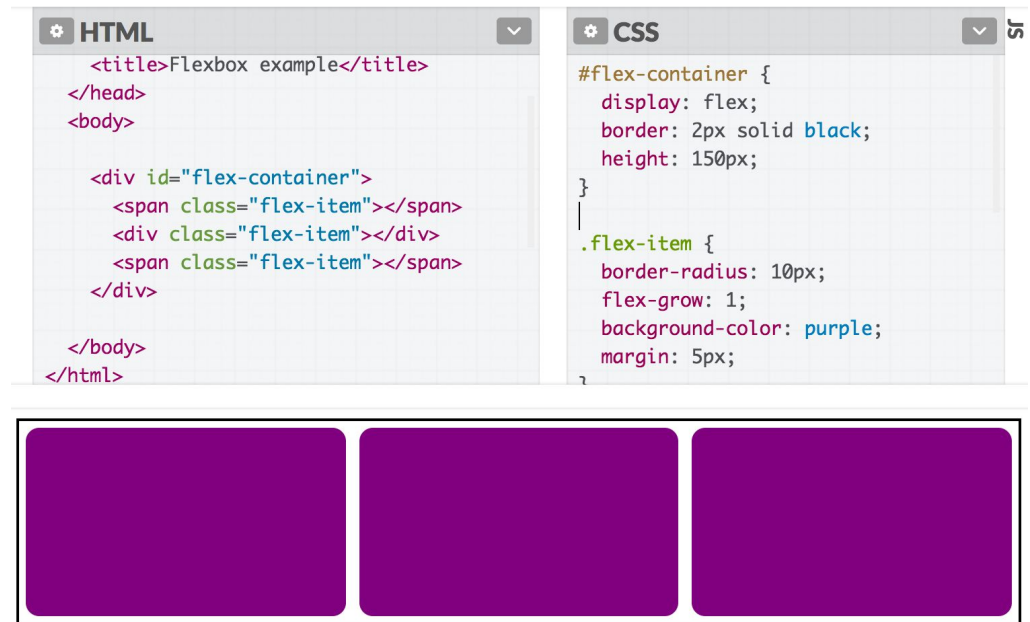
If we set `flex-grow: 1`, the flex items fill the empty space:



Flex item height**?!

Note that **flex-grow** only controls width*.

So why does the height** of the flex items seem to "grow" as well?

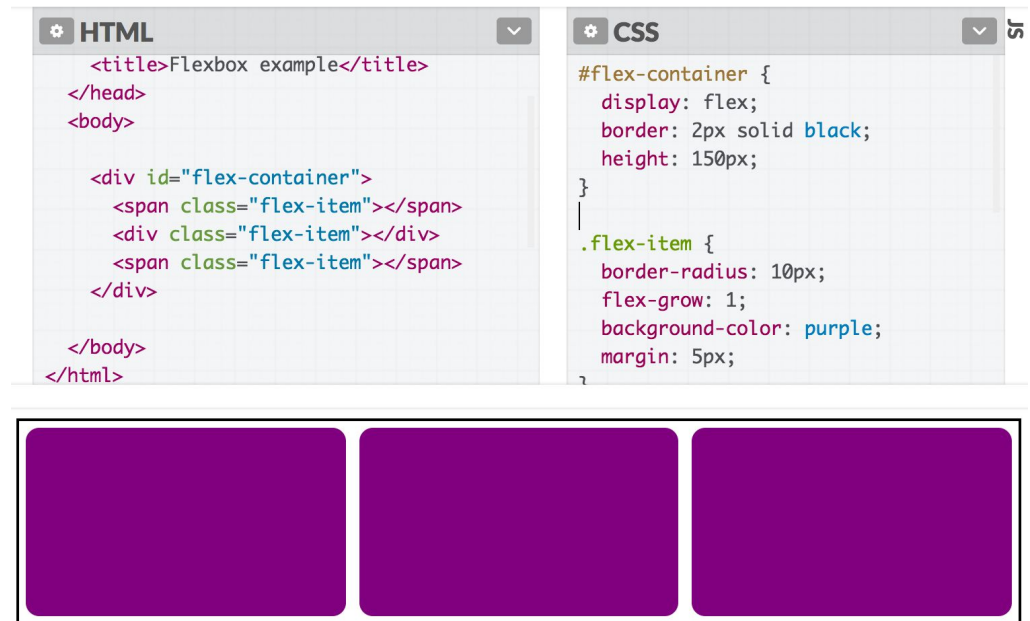


*width in the case of rows; height in the case of columns

**height in the case of rows; width in the case of columns

align-items: stretch;

The default value of align-items is stretch, which means every flex item grows vertically* to fill the container by default.



*vertically in the case of rows;
horizontally in the case of columns

align-items: stretch;

If we set another value for `align-items`, the flex items disappear again because the height is now content height, which is 0:



The screenshot shows a code editor with two panels: HTML and CSS. The HTML panel contains the following code:

```
<title>Flexbox example</title>
</head>
<body>

  <div id="flex-container">
    <span class="flex-item"></span>
    <div class="flex-item"></div>
    <span class="flex-item"></span>
  </div>

</body>
</html>
```

The CSS panel contains the following code:

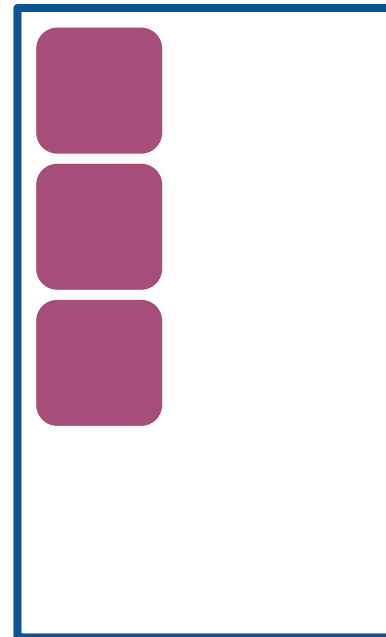
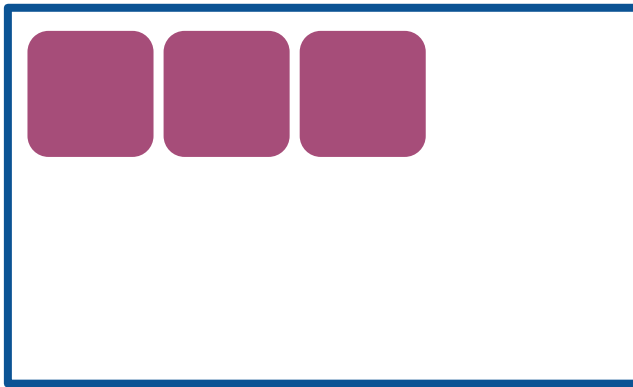
```
#flex-container {
  display: flex;
  align-items: flex-start;
  border: 2px solid black;
  height: 150px;
}

.flex-item {
  border-radius: 10px;
  flex-grow: 1;
  background-color: purple;
  margin: 5px;
}
```



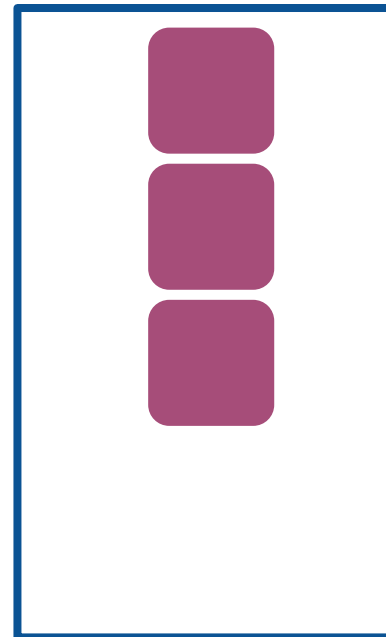
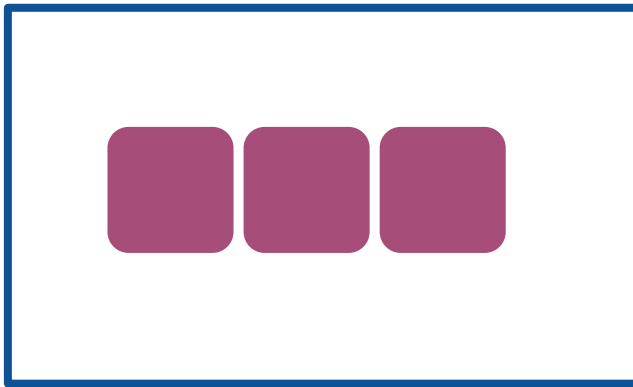
Flex layout recap

- If you set `display: flex`, the element is now a **flex container** and its direct children are **flex items**.
- The items in a flex container will layout in a row or column depending on the `flex-direction` of the container.



Flex layout recap

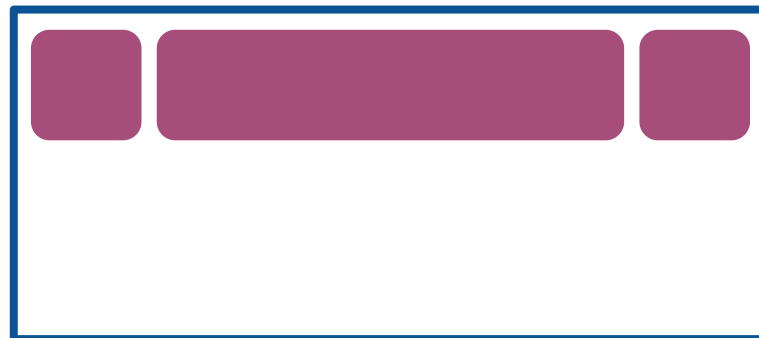
- **justify-content** distributes the items horizontally for flex-direction: row, vertically for column
- **align-items** distributes the items vertically for flex-direction: row, horizontally for column



Flex layout recap

For `flex-direction: row`:

- The **flex basis** is the initial width of a flex item
 - This is either the explicitly set width, the explicitly set `flex-basis`, or the content width
- The width of a flex item will **shrink** to fit the container if `flex-shrink` is set to 1 (disabled if 0)
- The width of a flex item will **grow** to fit the remaining space if `flex-grow` is set to 1 (disabled if 0)



Flex layout recap

For `flex-direction: row`:

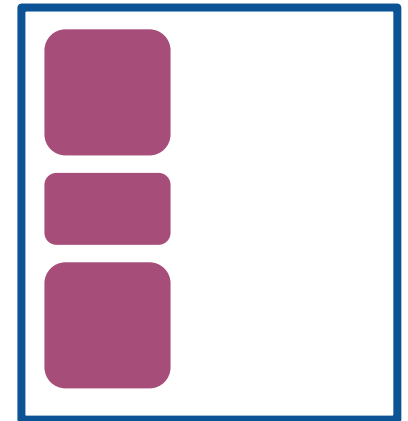
- The height of a flex item is either:
 - the explicitly set height on the item, or
 - the content height on the item, or
 - the height of the container if the container's `align-items: stretch`;



Flex layout recap

For flex-direction: column:

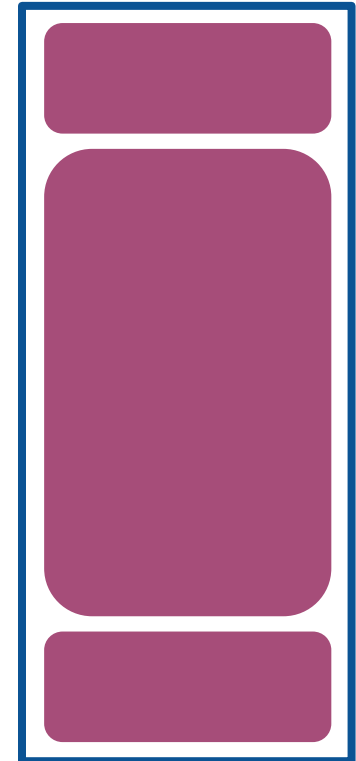
- The **flex basis** is the initial height of a flex item
 - This is either the explicitly set height, the explicitly set flex-basis, or the content height
- The height of a flex item will **shrink** to fit the container if flex-shrink is set to 1 (disabled if 0)
- The height of a flex item will **grow** to fit the remaining space if flex-grow is set to 1 (disabled if 0)



Flex layout recap

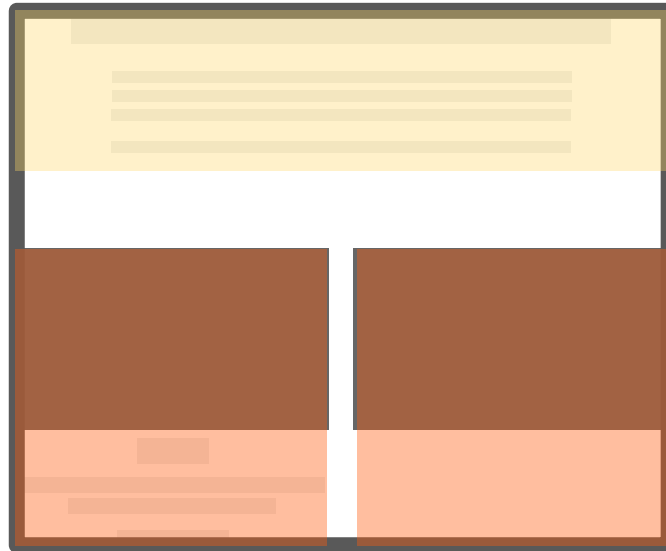
For `flex-direction: column`:

- The width of a flex item is either:
 - the explicitly set `width` on the item,
or
 - the content width on the item,
or
 - the width of the container if the
container's `align-items`:
`stretch`;



That's still just scratching the surface
of flex box...

...but we now know enough to
continue our layout!



Random useful CSS

calc

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

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

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);
```



([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;
```

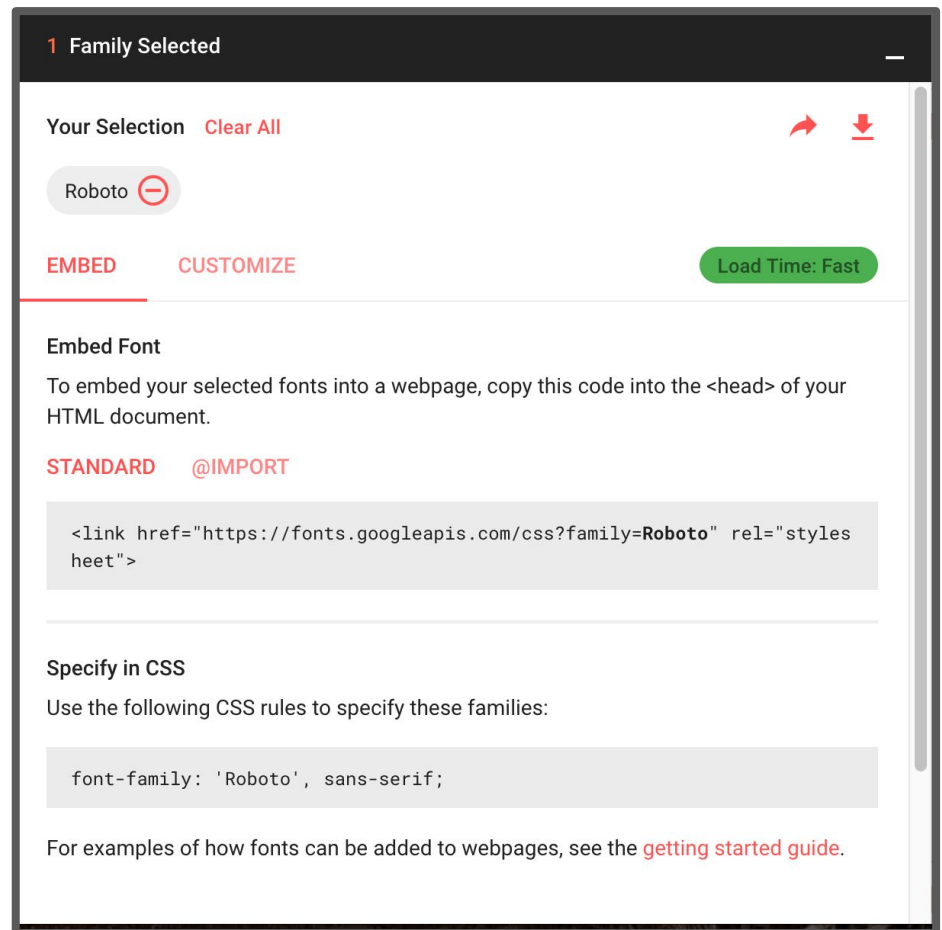
([CodePen](#): Try resizing the window)

Web Fonts

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

The instructions are pretty self-explanatory.

You can also load a totally custom font via [font-face@](#) (which we won't go over in class).



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.