Web Programming Fundamentals

February 2019

These slides are based on Victoria Kirst (vrk@stanford.edu) course CS193X. You can equally check out here (https://cs193x.stanford.edu) for the original course materials

Today's schedule

Today

- Squarespace Layout
  - Single row/column flexbox
- `vh / vw / box-sizing`

Friday

- `position`
- Random helpful CSS
- Mobile layouts
- CSS wrap-up

Monday

- Intro to JavaScript

Announcements

🚨 **Homework 1 deadline extended!** 🚨

- Due ~~Mon Apr 17~~ **Wed Apr 19!**
- Details here

Homework 2 will go out Wed Apr 19 as well.
See syllabus for adjusted schedule.

Victoria's Office Hours --> Friday

- Due to a meeting, my office hours will be Friday after class this week instead of today.

Amy / Cindy's Office Hours canceled this afternoon

- Email if you want to meet me at 4 in my office

Mistake on padding/margin

The shorthand for padding and margin actually go clockwise, not counter-clockwise (which...makes more sense)

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

(Previous slides now fixed)

Font-related CSS review

Name	Description
`font-family`	Font face (mdn)
`color`	Font color (and always font color) (mdn)
`font-size`	Font size (mdn)
`line-height`	Line height (mdn)
`text-align`	Alignment of text (mdn)

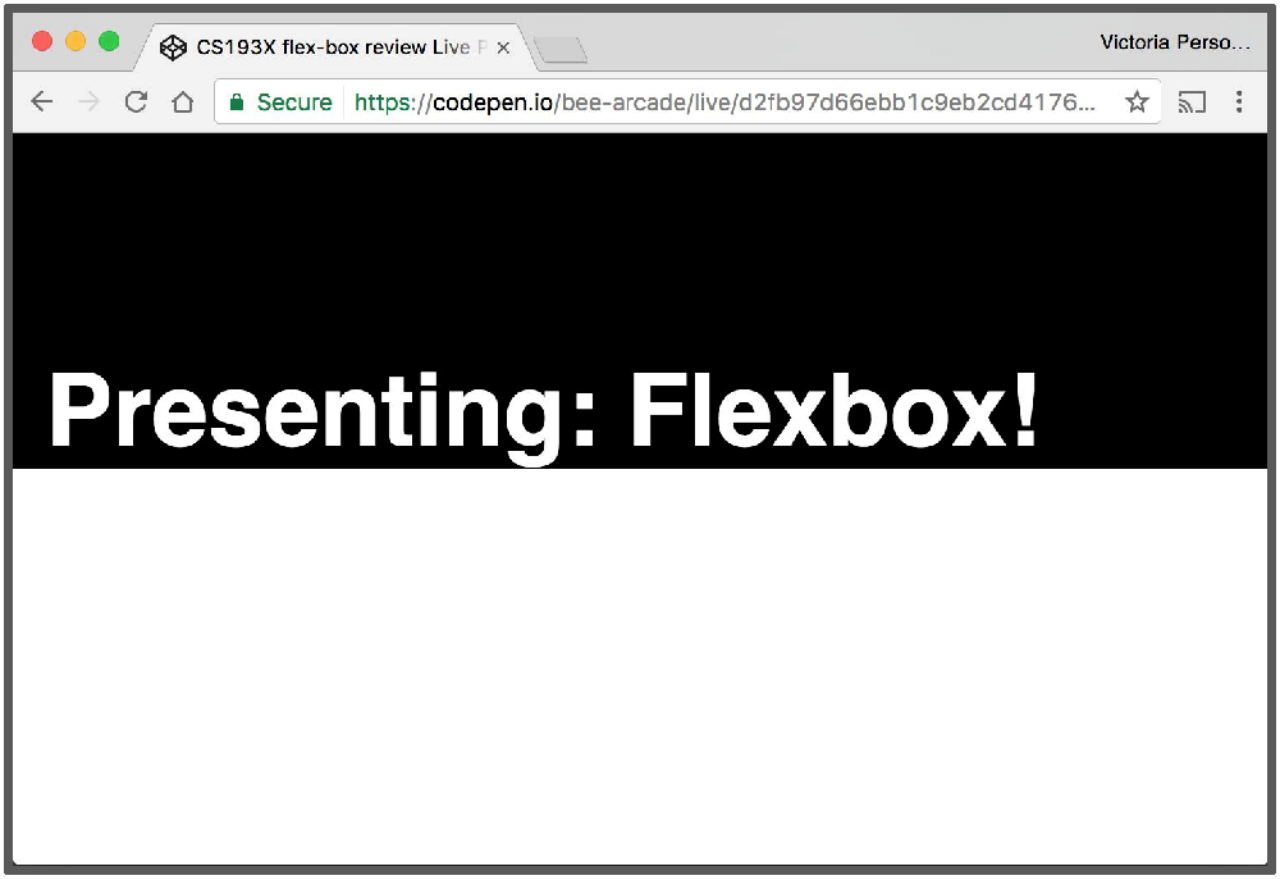
More font-related CSS

Name	Description
`text-decoration`	Can set `underline`, `line-through` (strikethrough) or `none` (e.g. to unset underline on hyperlinks) (mdn)
`text-transform`	Can change font **case**, i.e. `uppercase`, `lowercase`, `capitalize`, `none` (mdn)
`font-style`	Can set to `italic` or `normal` (e.g. to unset italic on `<em>`) (mdn)
`font-weight`	Can set to `bold` or `normal` (e.g. to unset bold on `h1` - `h6`) (mdn)
`letter-spacing`	Controls the space between letters (mdn)

Flexbox

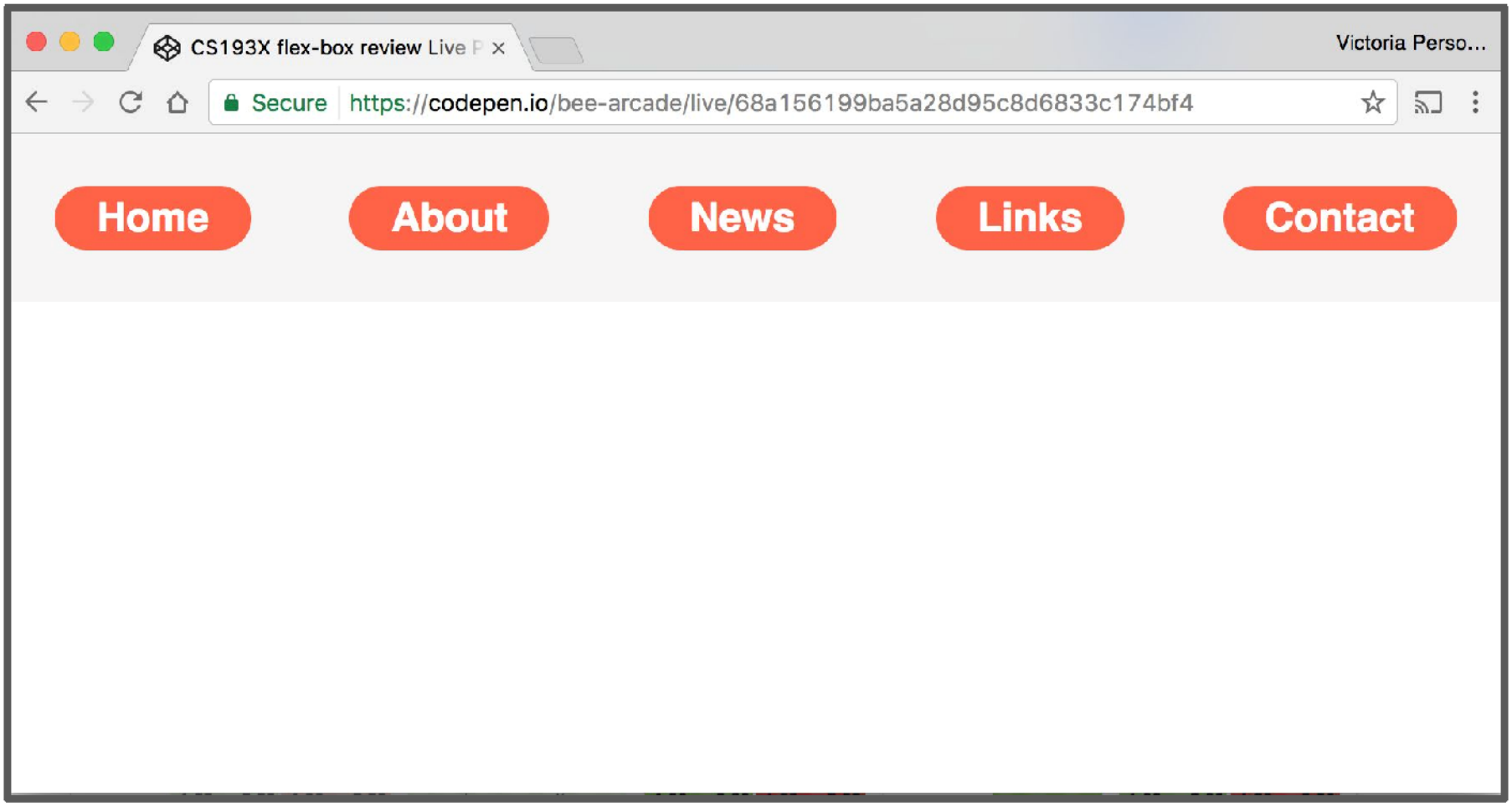
Review: Flexbox

How do we create this look? (Codepen)



Review: Flexbox

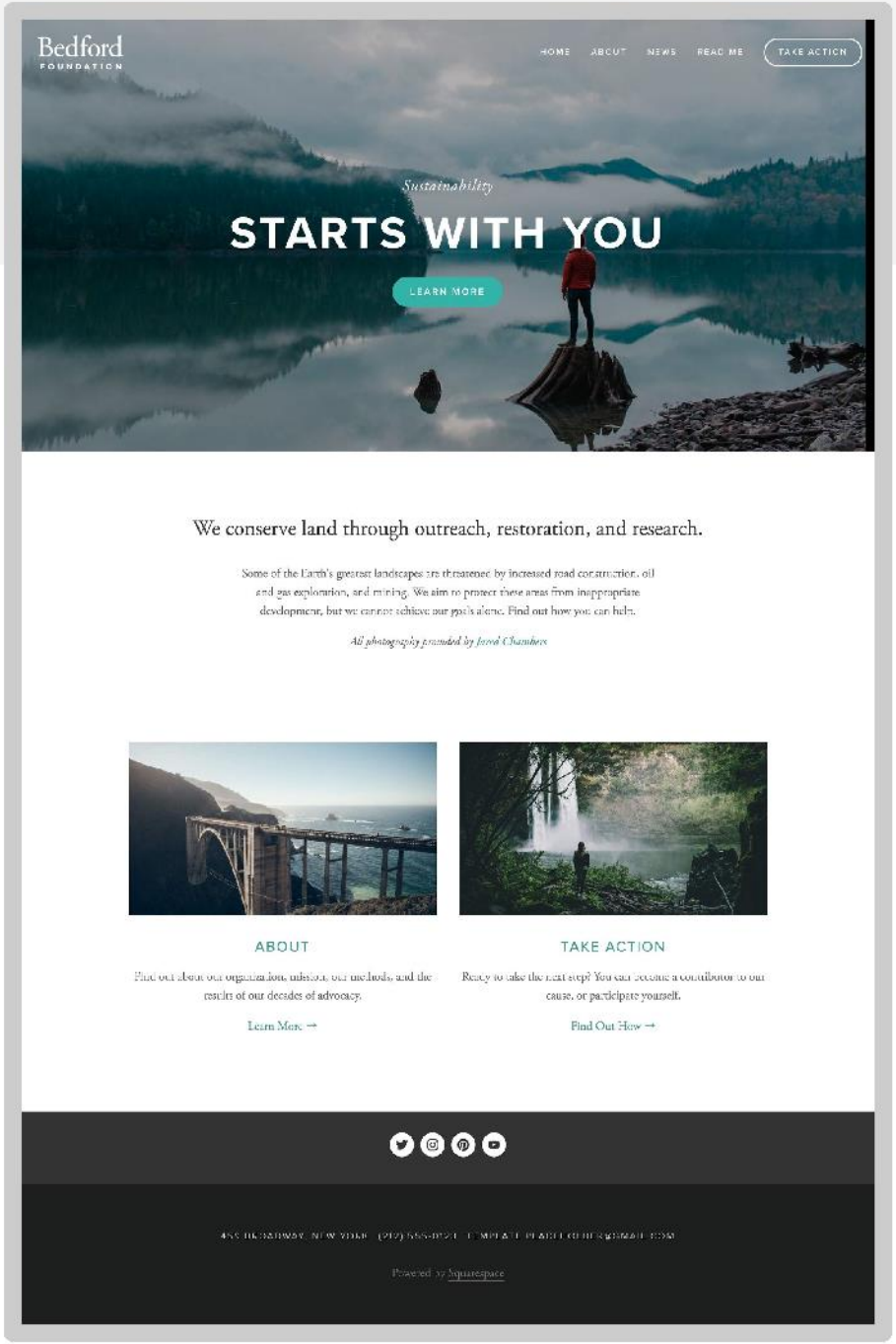
How do we create this look? (Codepen)



Continuing where we left off!

Goal

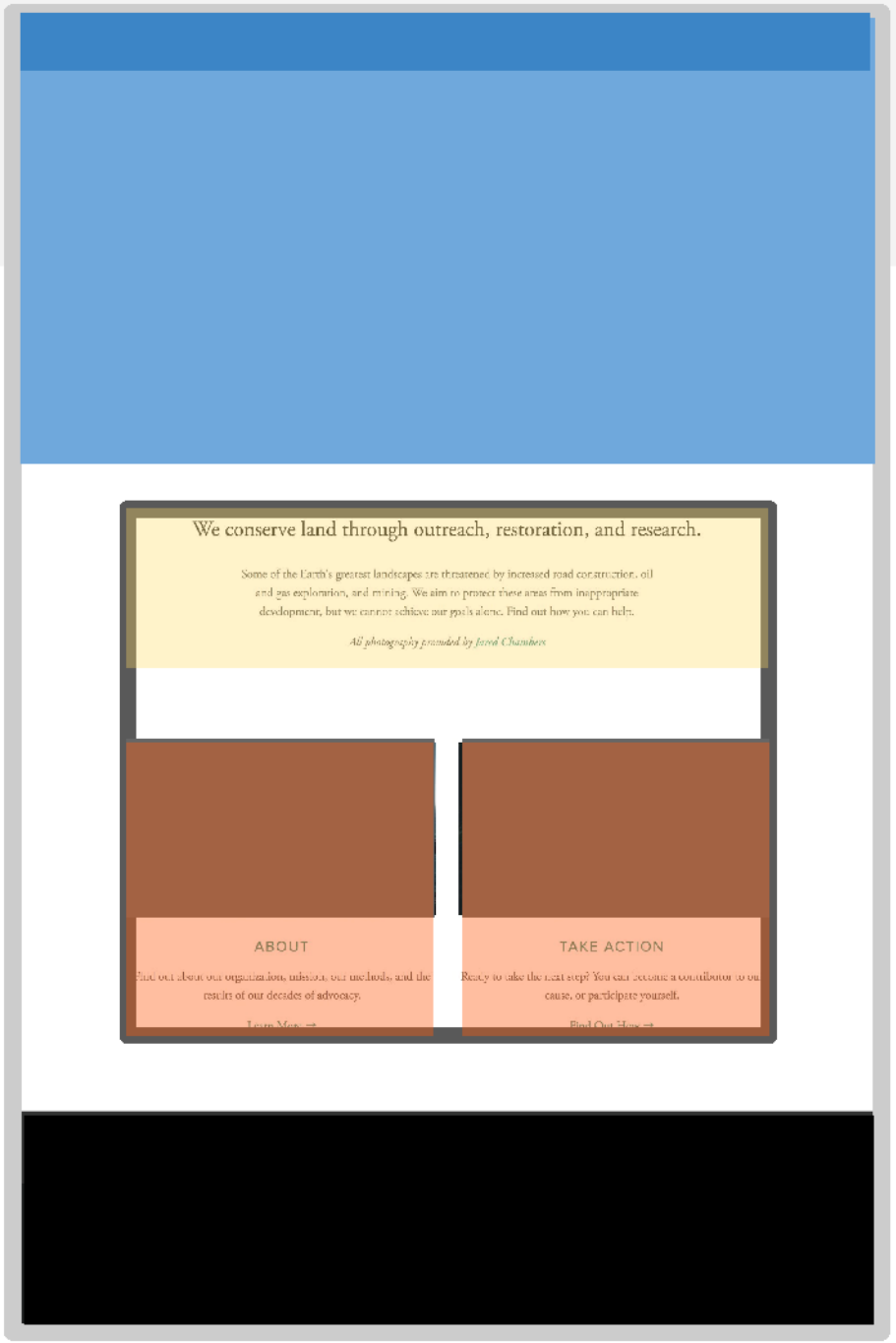
We were trying to create a layout that looks sort of like this:



Status

We broke up the layout into a bunch of colored boxes:

And we got kind of stuck trying to position the orange boxes.



Recall: block layouts

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

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>
</html>
```

CSS

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

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

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

(Review block and inline!)

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>
</html>
```

CSS

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

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

(Please make sure you completely understand why the elements do not show up!)

Check out **block vs inline guide**

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;
}
```

???

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

Flex layouts

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;
}
```

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.

Follow along on CodePen

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

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**.

The explicit width* of a flex item is respected *for all flex items*, regardless of whether the flex item is inline, block, or inline-block.

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

Flex basis

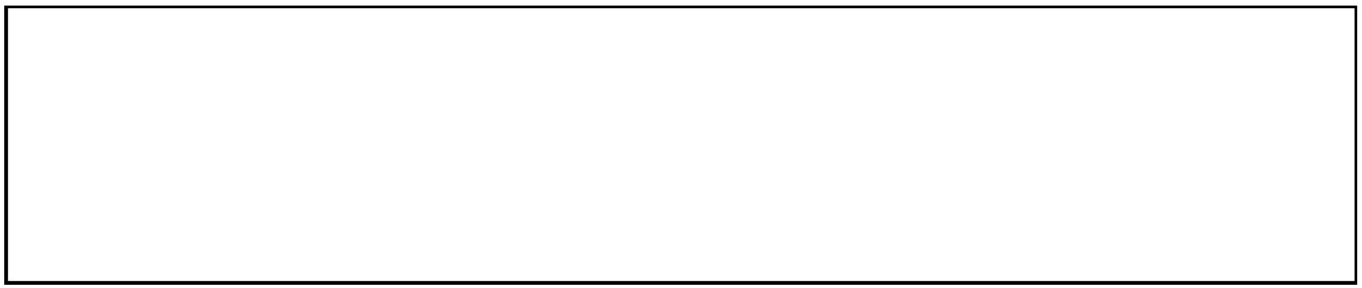
If we unset the `height` and `width`, our flex items disappears, because the **flex basis** is now the content size, which is empty:

HTML

```
    <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-shrink`

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

`flex-shrink`:

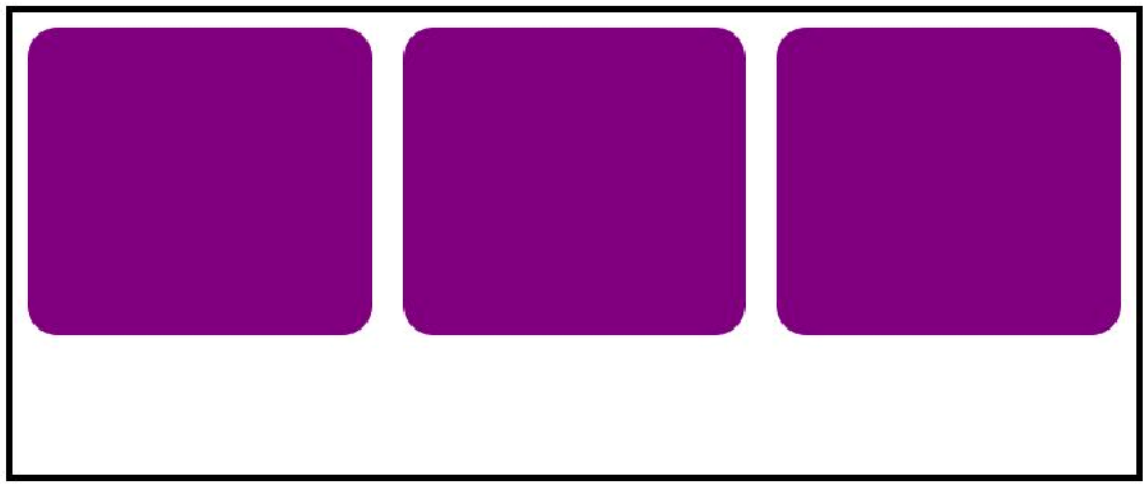
- If set to 1, the flex item shrinks itself as small as it can in the space available.
- If set to 0, the flex item does not shrink.

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

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

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

```css
.flex-item {
  width: 500px;
  height: 100px;

  border-radius: 10px;
  background-color: purple;
  margin: 5px;
}
```

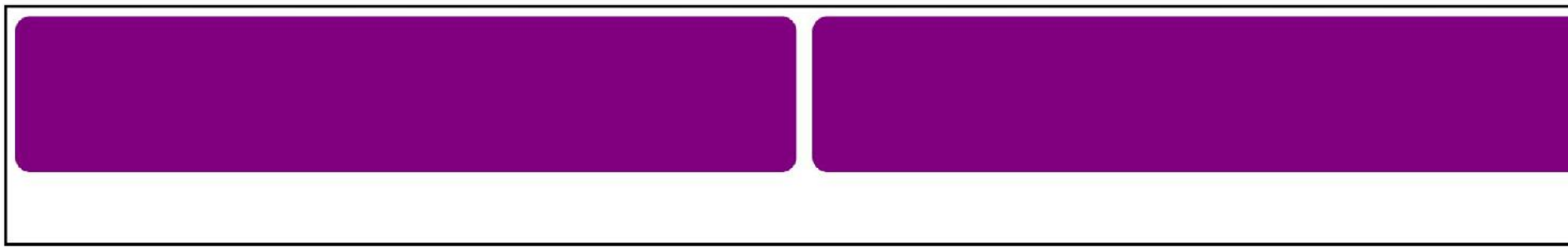


The flex items' widths all shrink to fit within the container.

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

```css
.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

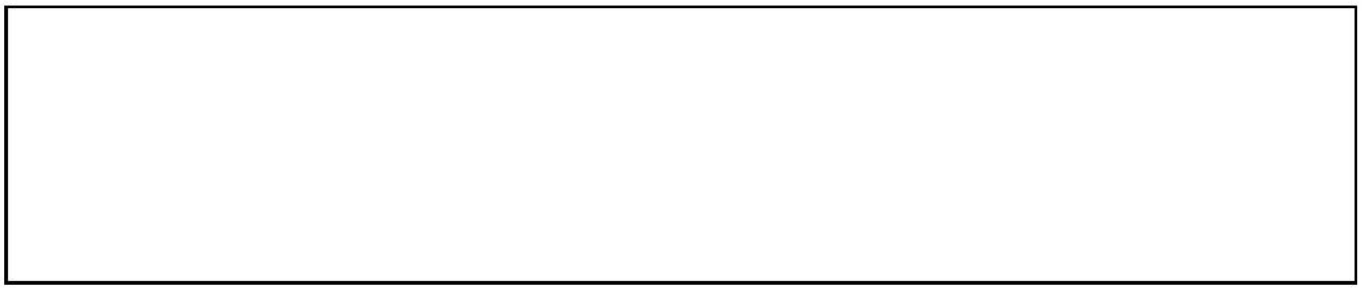
Let's unset the height and width of our flex items again:

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

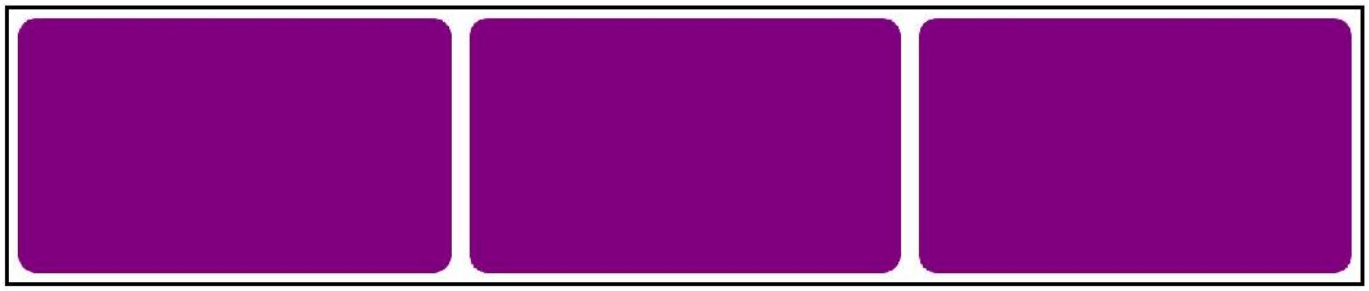
If we set `flex-grow: 1`, the flex items fill the empty space:

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;
  flex-grow: 1;
  background-color: purple;
  margin: 5px;
}
```



Flex item height**?!

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

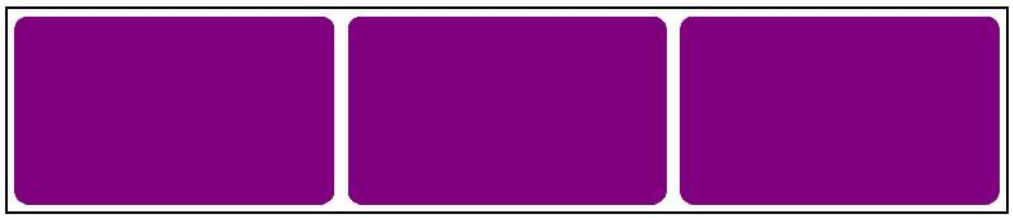
So why does the height** of the flex items seem to "grow" as well?

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;
  flex-grow: 1;
  background-color: purple;
  margin: 5px;
}
```



*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.

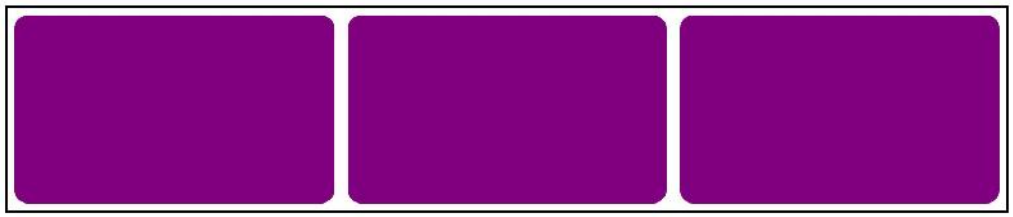
(This will not happen if the `height` on the flex item is set)

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;
  flex-grow: 1;
  background-color: purple;
  margin: 5px;
}
```



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

`align-items: stretch;`

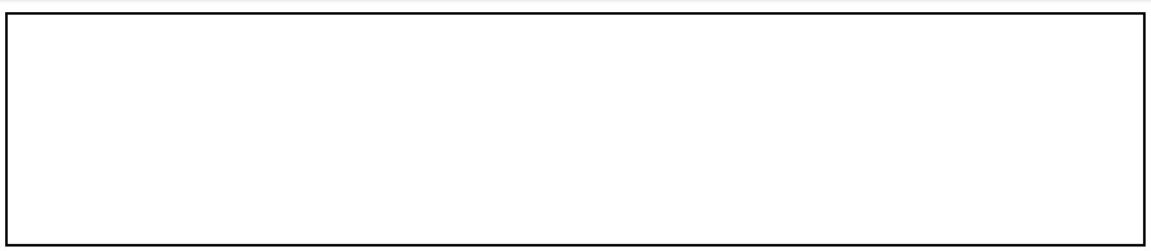
If we set another value for `align-items`, the flex items disappear again because the height is now content height, which is 0:

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>
```

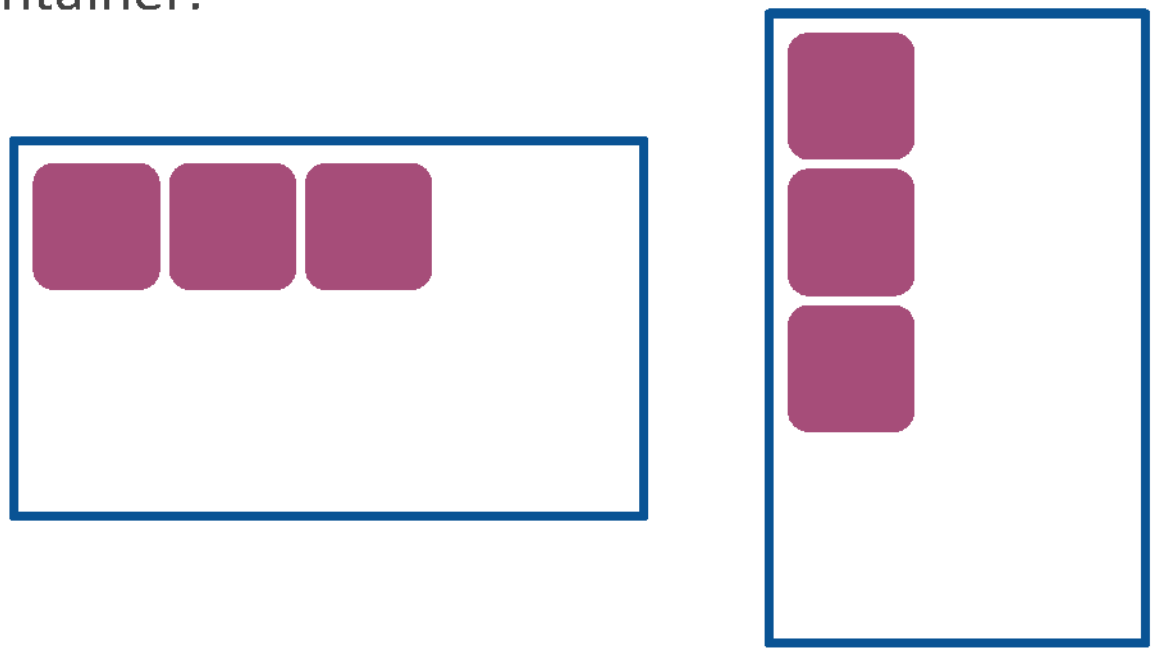
CSS

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

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



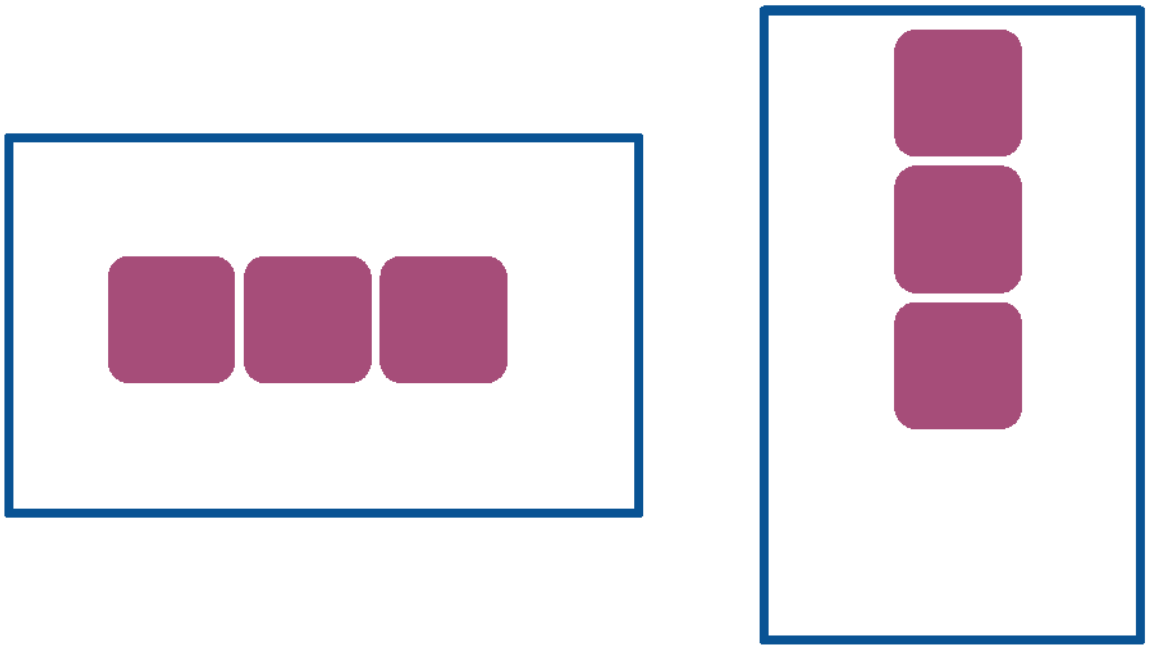
Flex layout recap

- 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-contents`** 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

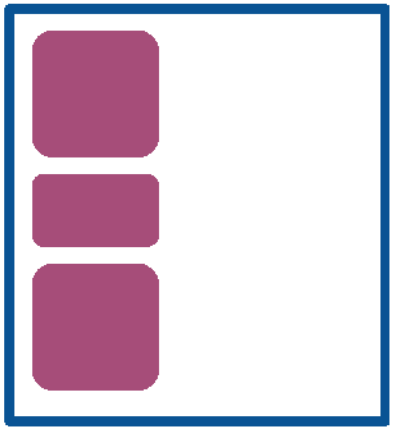
For `flex-direction: row`:

- 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`:

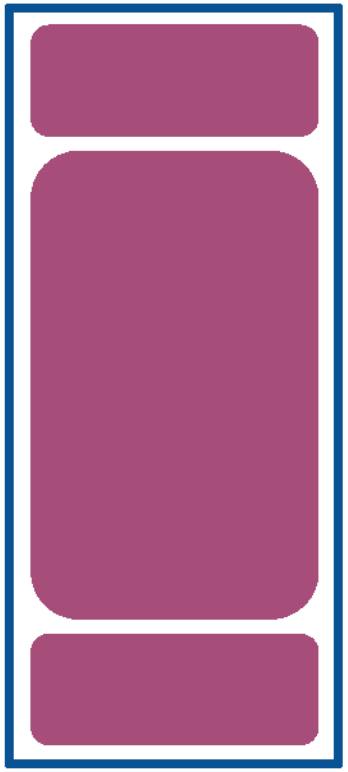
- The **flex basis** is the initial height of a flex item
  - This is either the explicitly set `height`, the explicitly set `flex-basis`, or the content height
- The height of a flex item will **shrink** to fit the container if `flex-shrink` is set to 1 (disabled if 0)
- The height of a flex item will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

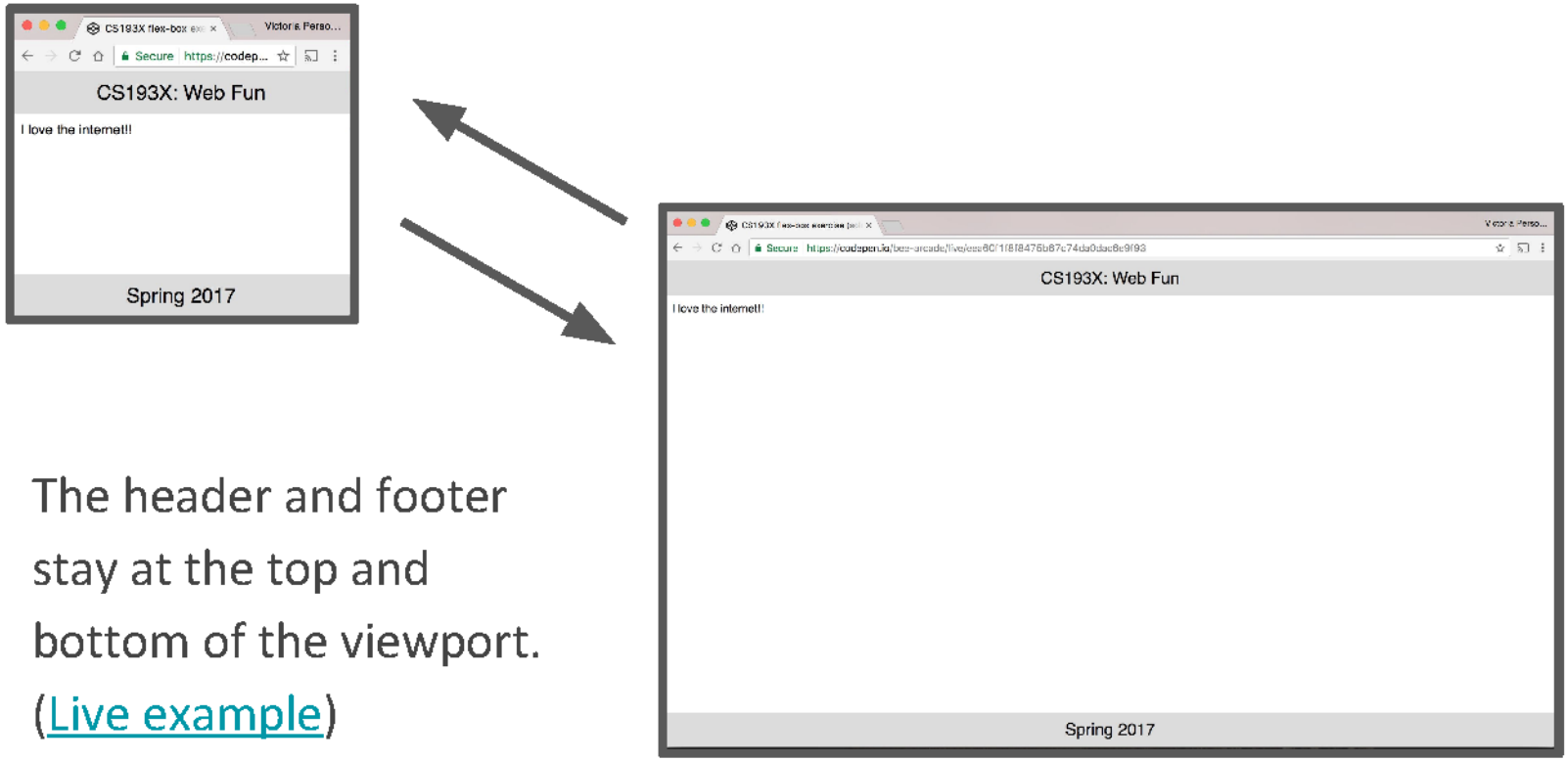
...but we now know enough to continue our layout!

Follow along on Codepen

Height and width quirks:
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)

`height` and `width` percentages

When `width` is defined as a percentage:

- `width` is specified as a percentage of the **containing block's** `width`.

When `height` is defined as a percentage:

- `height` is specified as a percentage of the **containing block's** `height`.

In other words, `height` and `width` are defined **relative to their parent element** when defined as a percentage.

height and width percentages

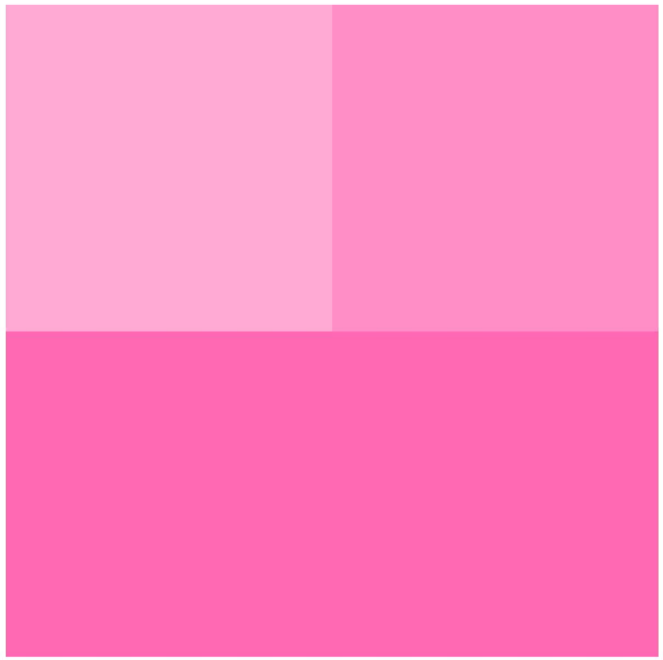
HTML

```
<div id="box">
  <div id="upper-half">
    <div id="upper-quarter"></div>
  </div>
</div>
```

CSS

```
#box {
  height: 500px;
  width: 500px;
  background-color: hotpink;
}

#upper-half {
  height: 50%;
  width: 100%;
}

#upper-quarter {
  height: 100%;
  width: 50%;
}

#box div {
 background-color: rgba(255, 255, 255, 0.25);
}
```

OUTPUT

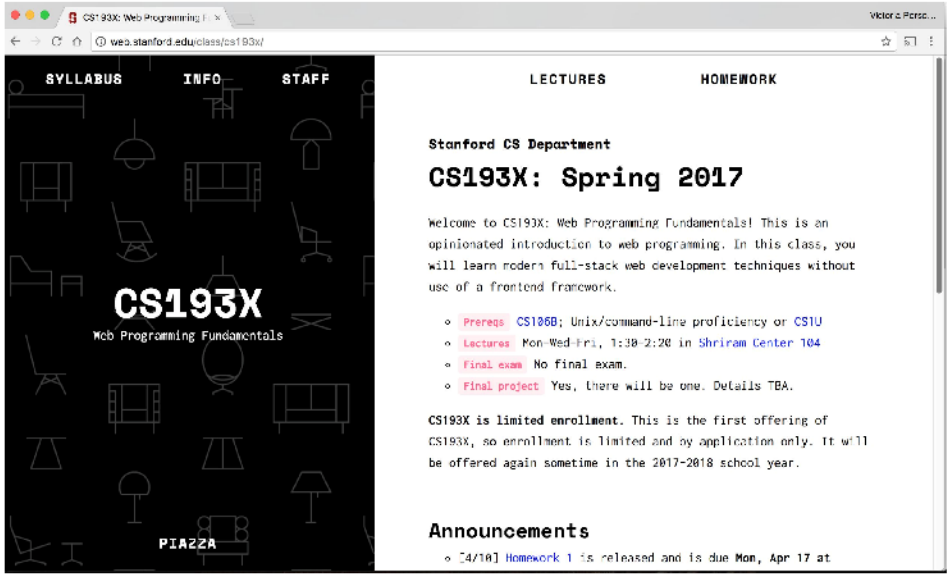


(Codepen)

Viewport?

Browser vocabulary:

- **viewport:** the rectangle where the webpage shows up, scrollable via a scrollbar
- **chrome:** all the UI that's *not* the webpage, i.e. everything but the viewport

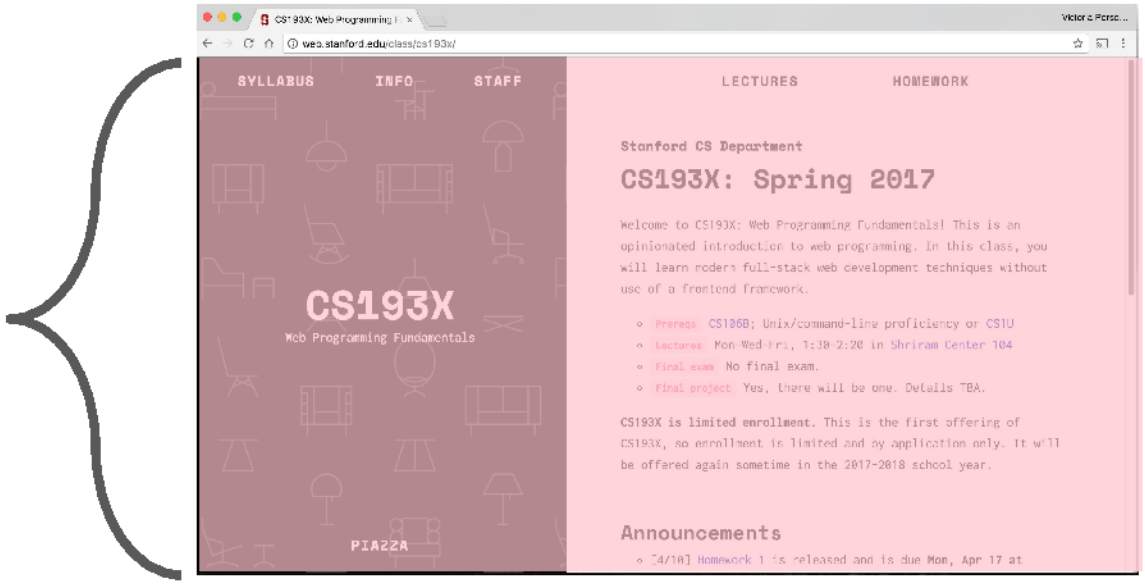


Viewport?

Browser vocabulary:

- **viewport:** the rectangle where the webpage shows up, scrollable via a scrollbar
- **chrome:** all the UI that's *not* the webpage, i.e. everything but the viewport

The viewport

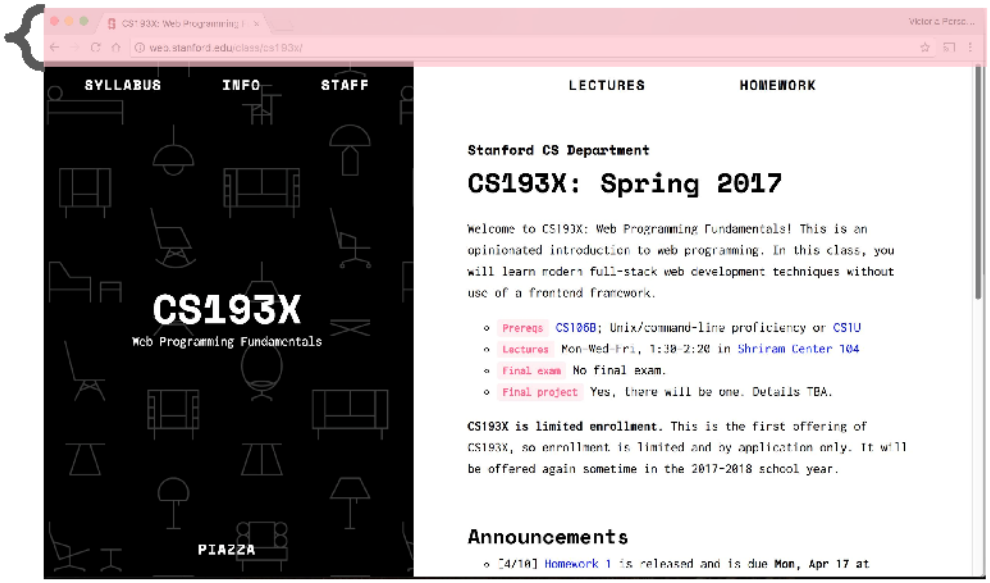


Viewport?

Browser vocabulary:

- **viewport:** the rectangle where the webpage shows up, scrollable via a scrollbar
- **chrome:** all the UI that's *not* the webpage, i.e. everything but the viewport

The chrome



vh and vw

You can define `height` and `width` in terms of the viewport

- Use units `vh` and `vw` to set `height` and `width` to the percentage of the viewport's height and width, respectively (mdn)
- `1vh` = 1/100th of the viewport height
- `1vw` = 1/100th of the viewport width

Example:

- `height: 100vh;`
- `width: 100vw;`

Flexbox example, solved

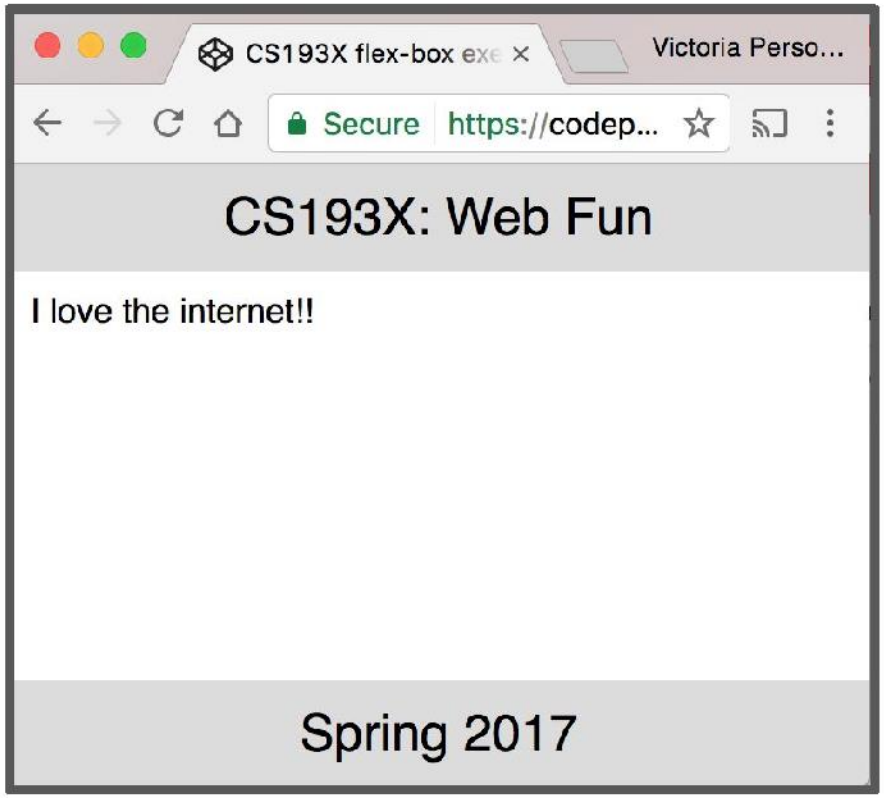
HTML

```
<article>
  <header>CS193X: Web Fun</header>
  <section>
    <p>I love the internet!!</p>
  </section>
  <footer>Spring 2017</footer>
</article>
```

CSS

```
article {
  display: flex;
  flex-direction: column;
  height: 100vh;
  width: 100%;
}

section {
  flex-grow: 1;
  padding: 10px;
}
```

(rest of the CSS)



(CodePen)

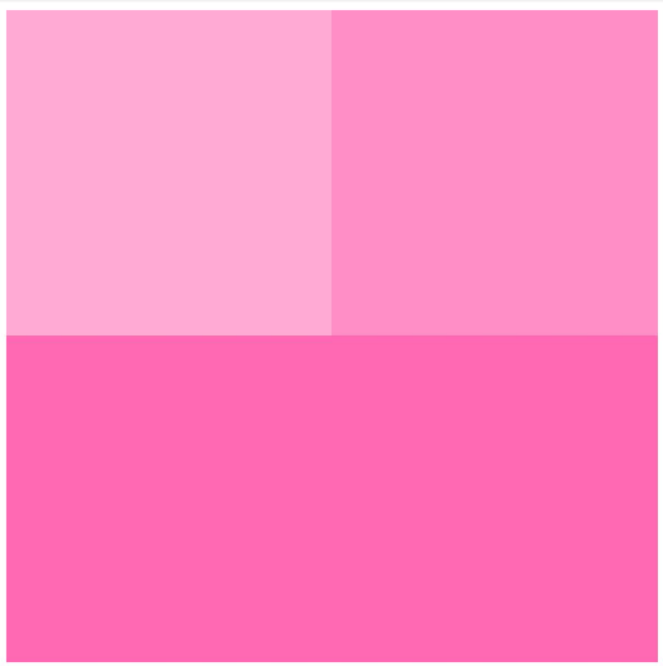
Aside: sizing

Q: What happens if we add a border to #upper-half?

```
<div id="box">
  <div id="upper-half">
    <div id="upper-quarter"></div>
  </div>
</div>
```

```
#upper-half {
  height: 50%;
  width: 100%;
}
```

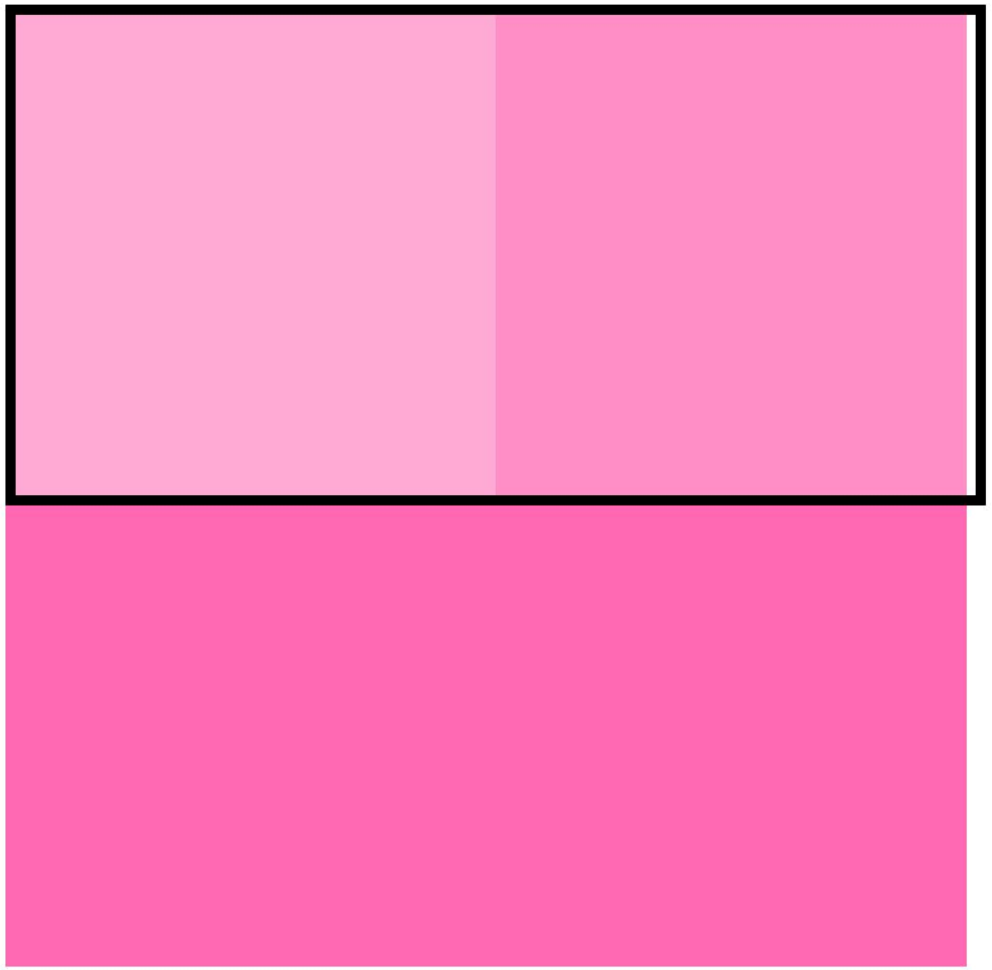
(rest of the css)



???

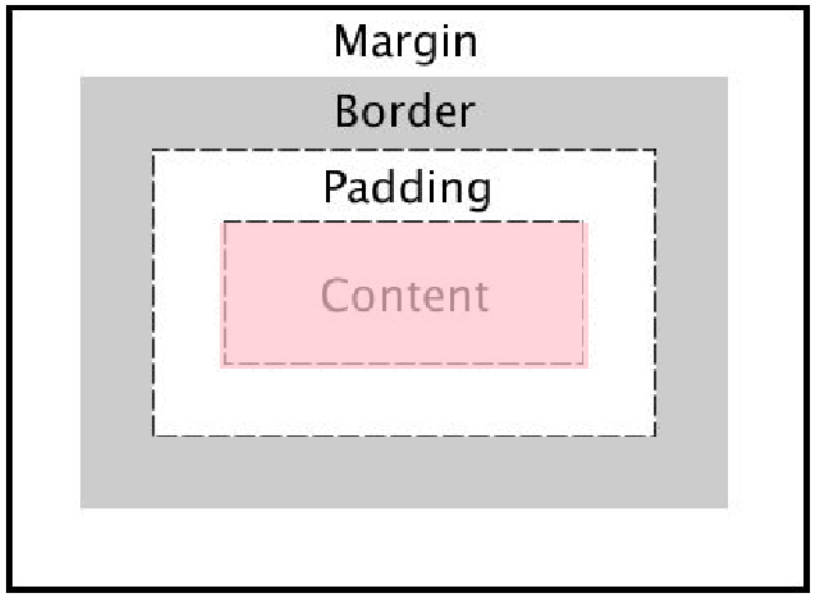
```
#upper-half {
  height: 50%;
  width: 100%;
  border: 5px solid black;
}
```

(rest of the CSS)



CSS box model width and height

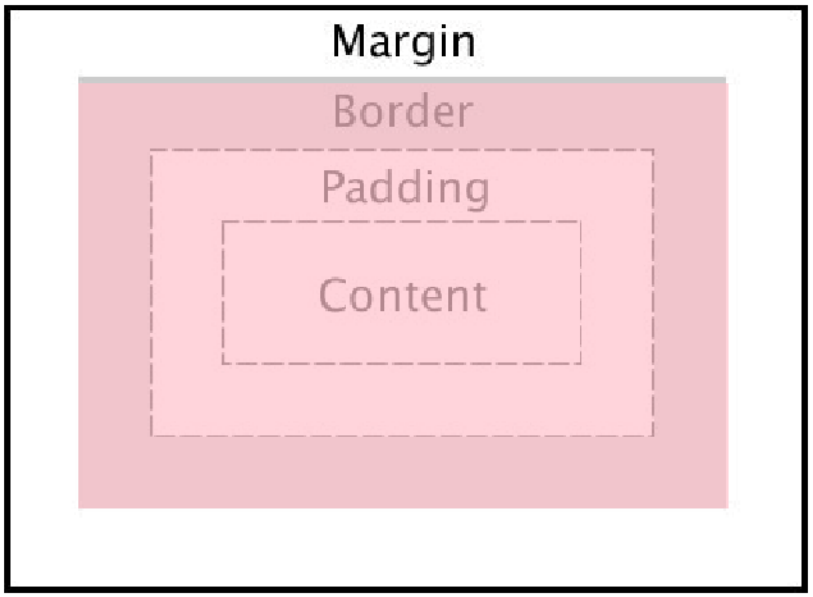
The box model defines CSS `width` and `height` properties to refer to the element's **content** width and height:



box-sizing

If you want to have `width` and `height` refer to the element's **border** width and height, use `box-sizing`:

- `box-sizing: border-box;`

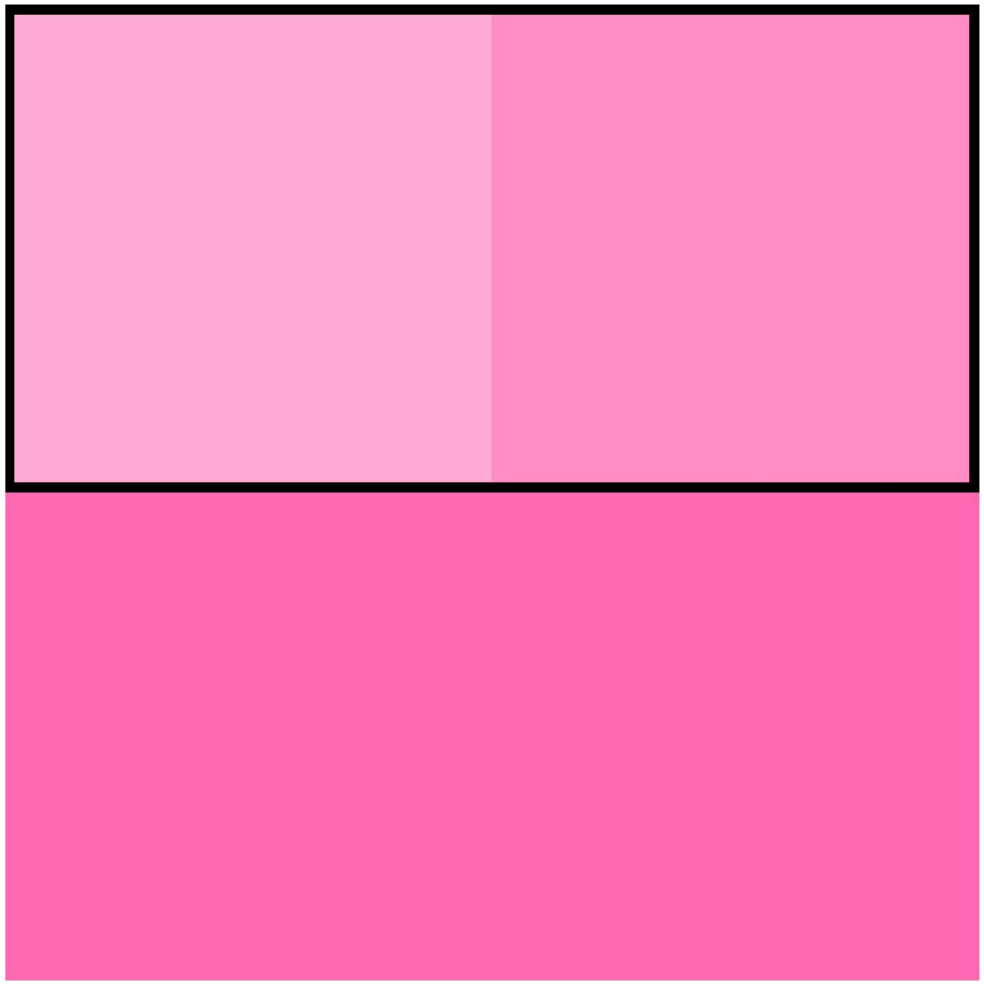


Note: Using `border-box` will include padding in the `width` and `height` as well.
Note: You **cannot** select padding-box or margin-box.

Fixed example

```css
#upper-half {
  height: 50%;
  width: 100%;
  border: 5px solid black;
  box-sizing: border-box;
}
```

(rest of the CSS)



Before we finish Squarespace...

Another rendering mode: `position`

Next time!