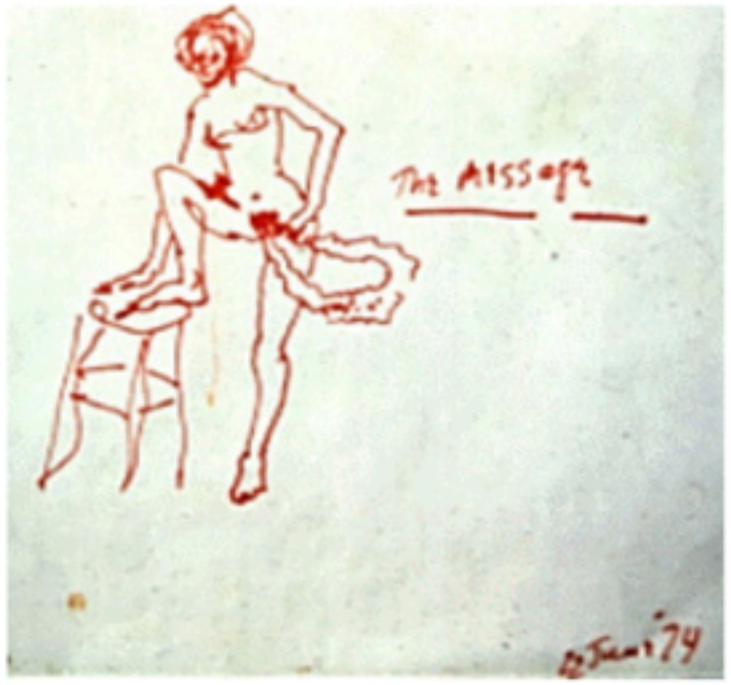
Interior Scroll



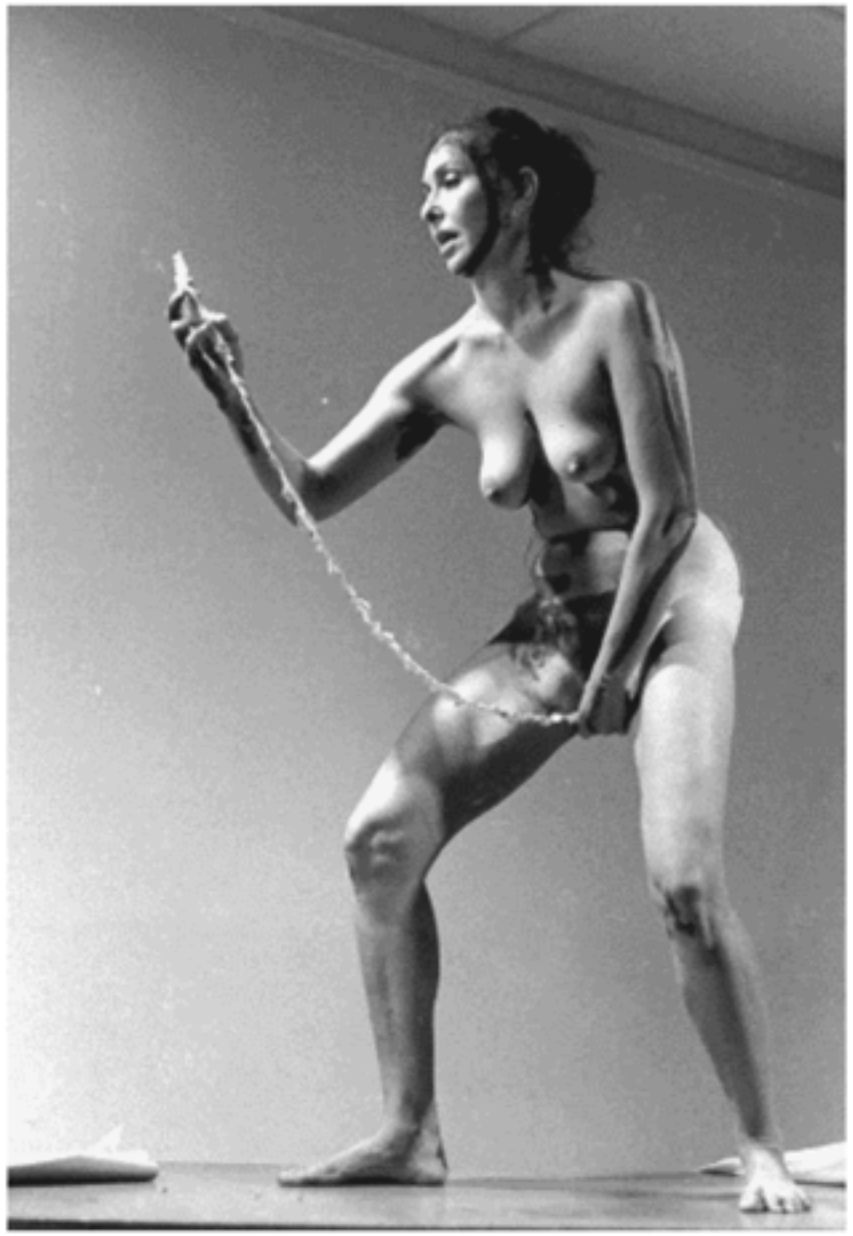
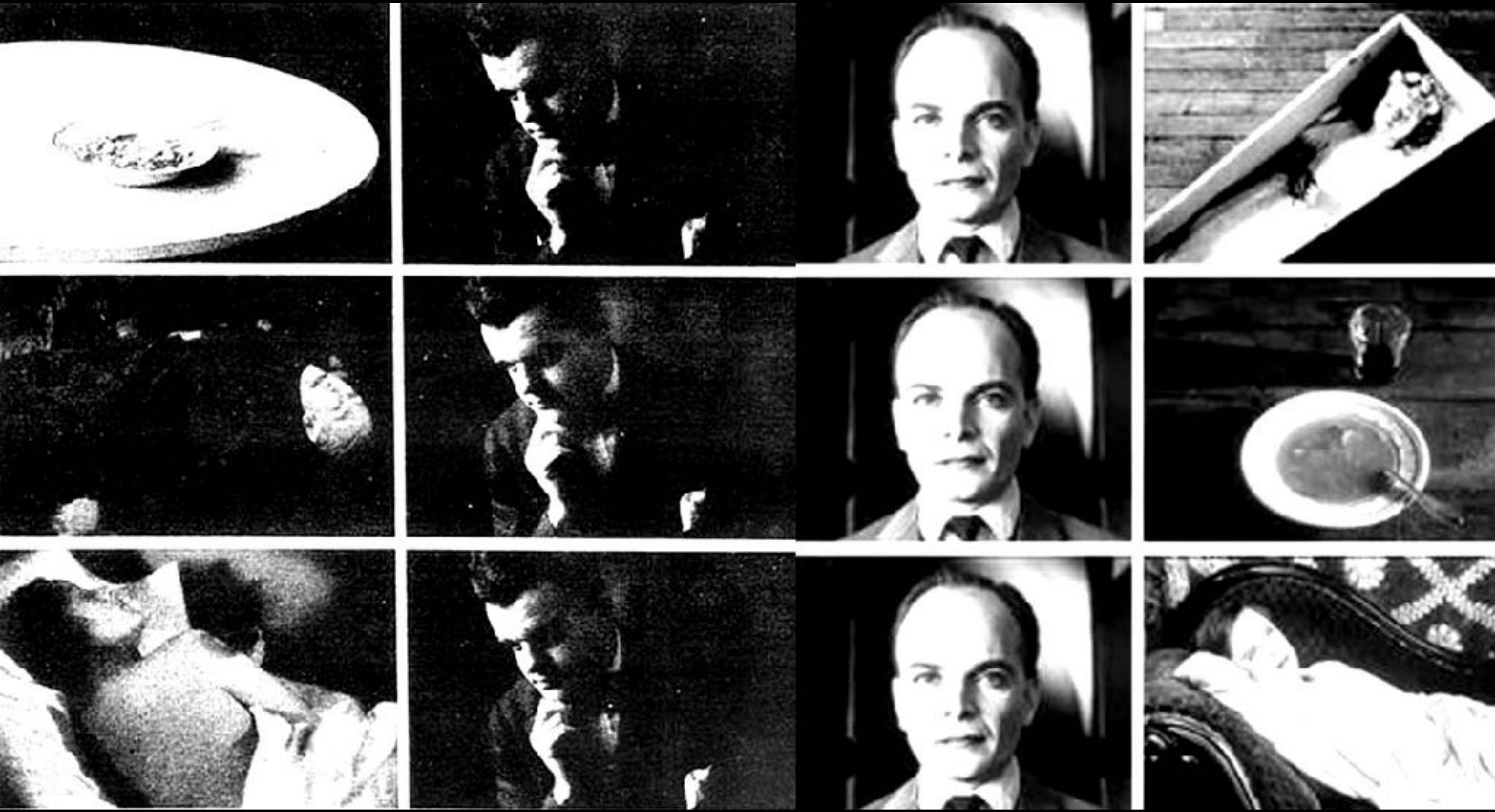


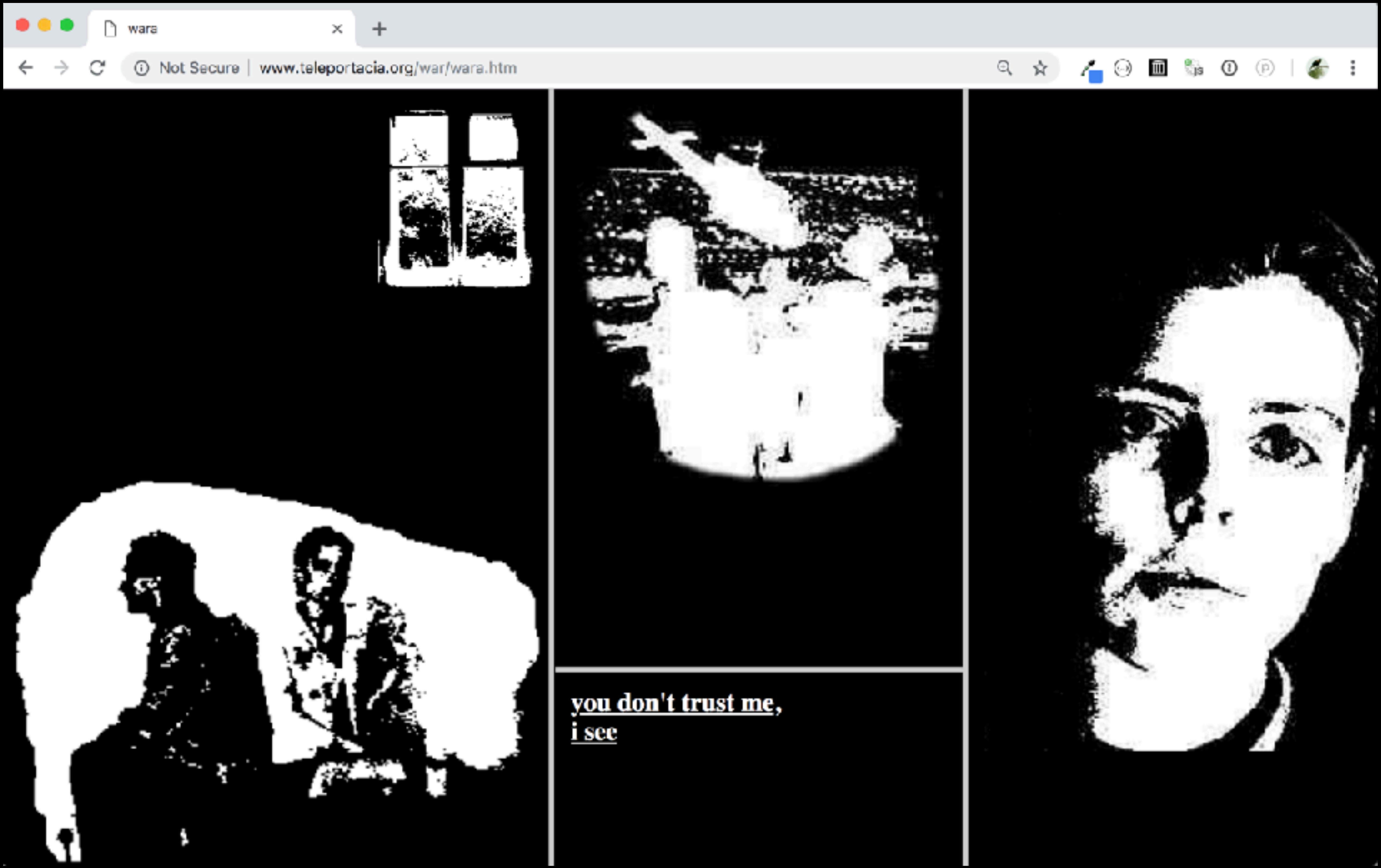
image from: [Carolee Schneemann's website]

Carolee Schneemann
Fuses, 1964 - 67

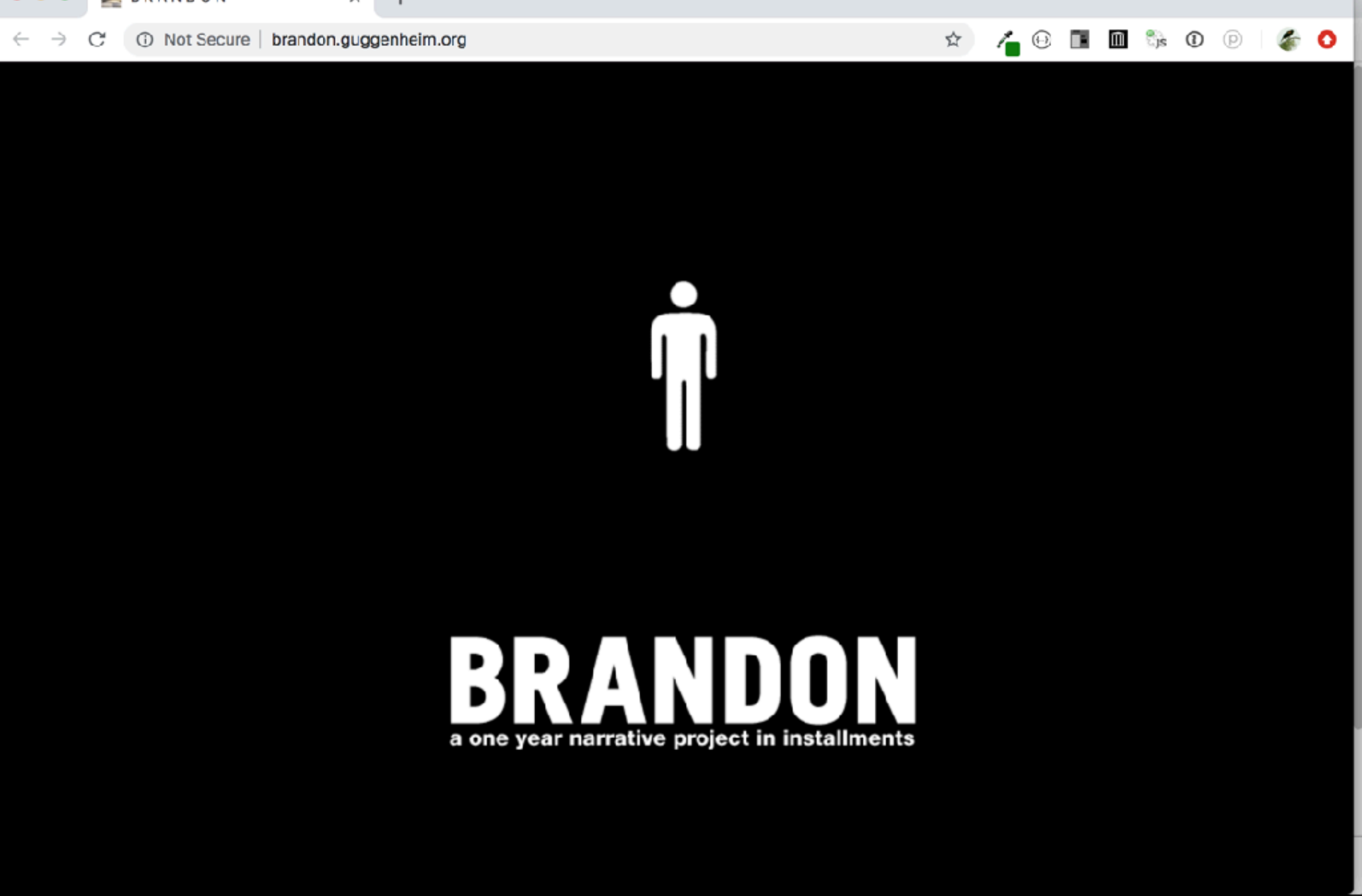
week 5 + 6: montage + ntwrks
(+ cyber feminism)



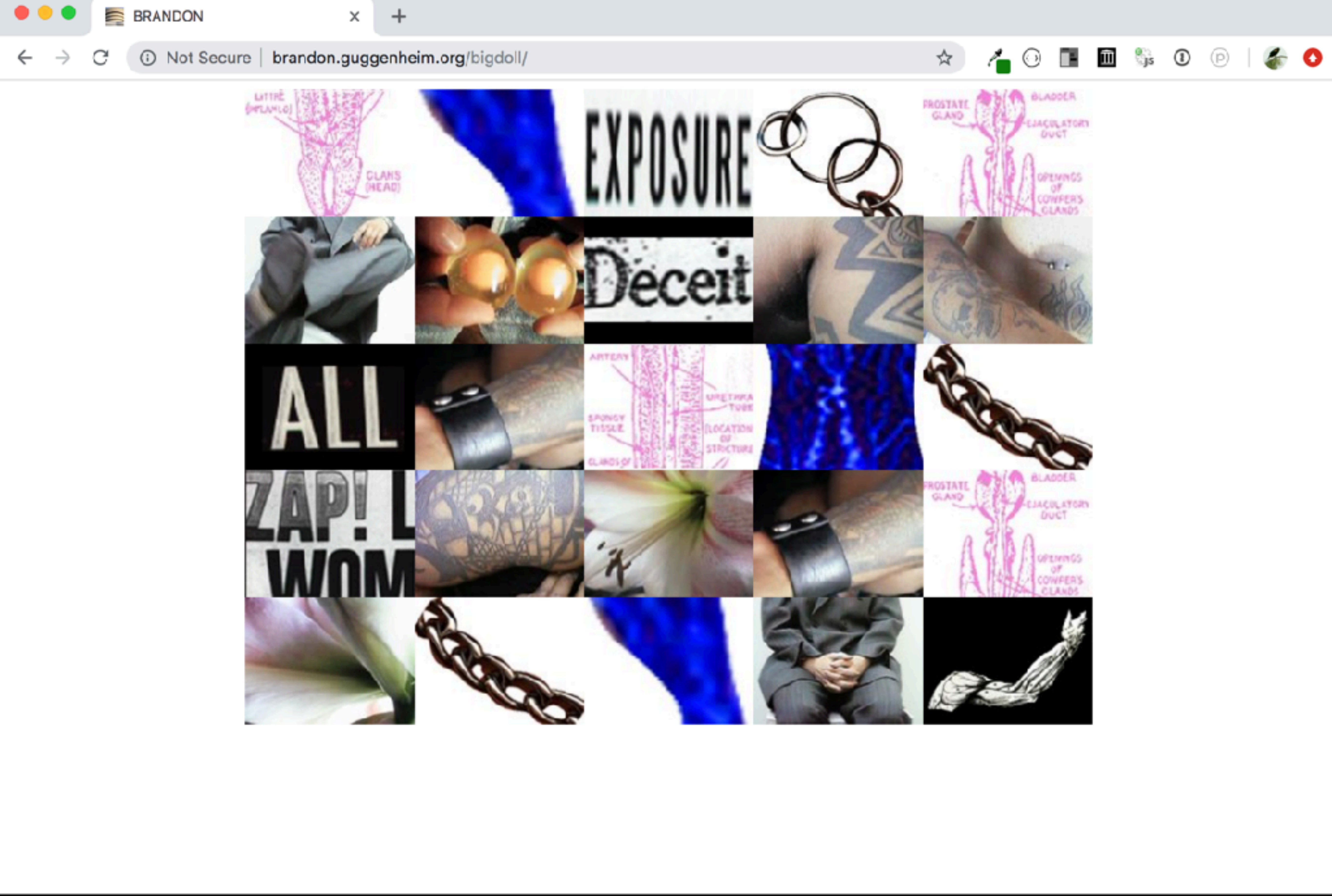
Lev Kuleshov, Russia
The Kuleshov Effect, 1929



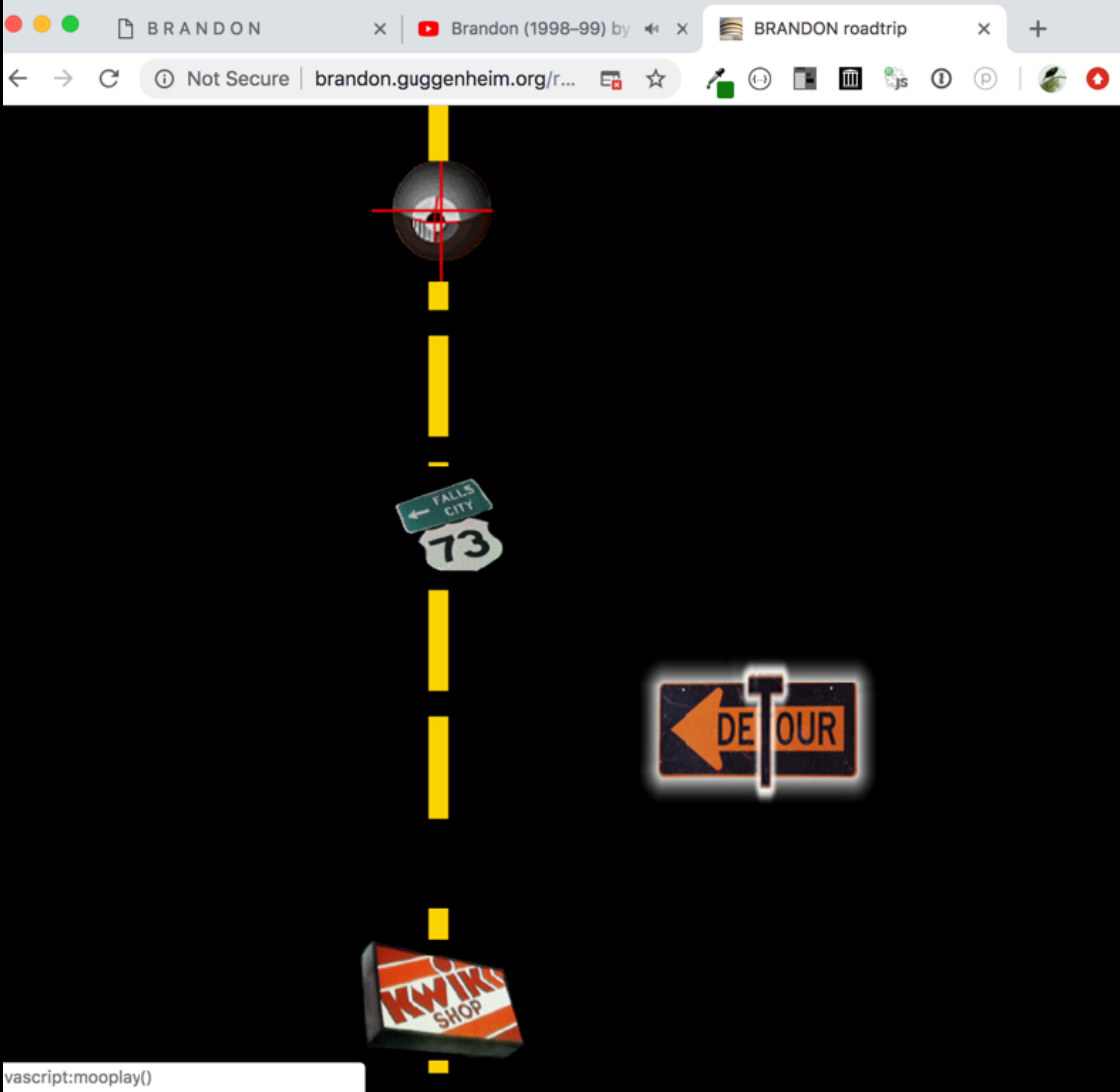
My Boyfriend Came Back From War, 1996
Olia Lialina



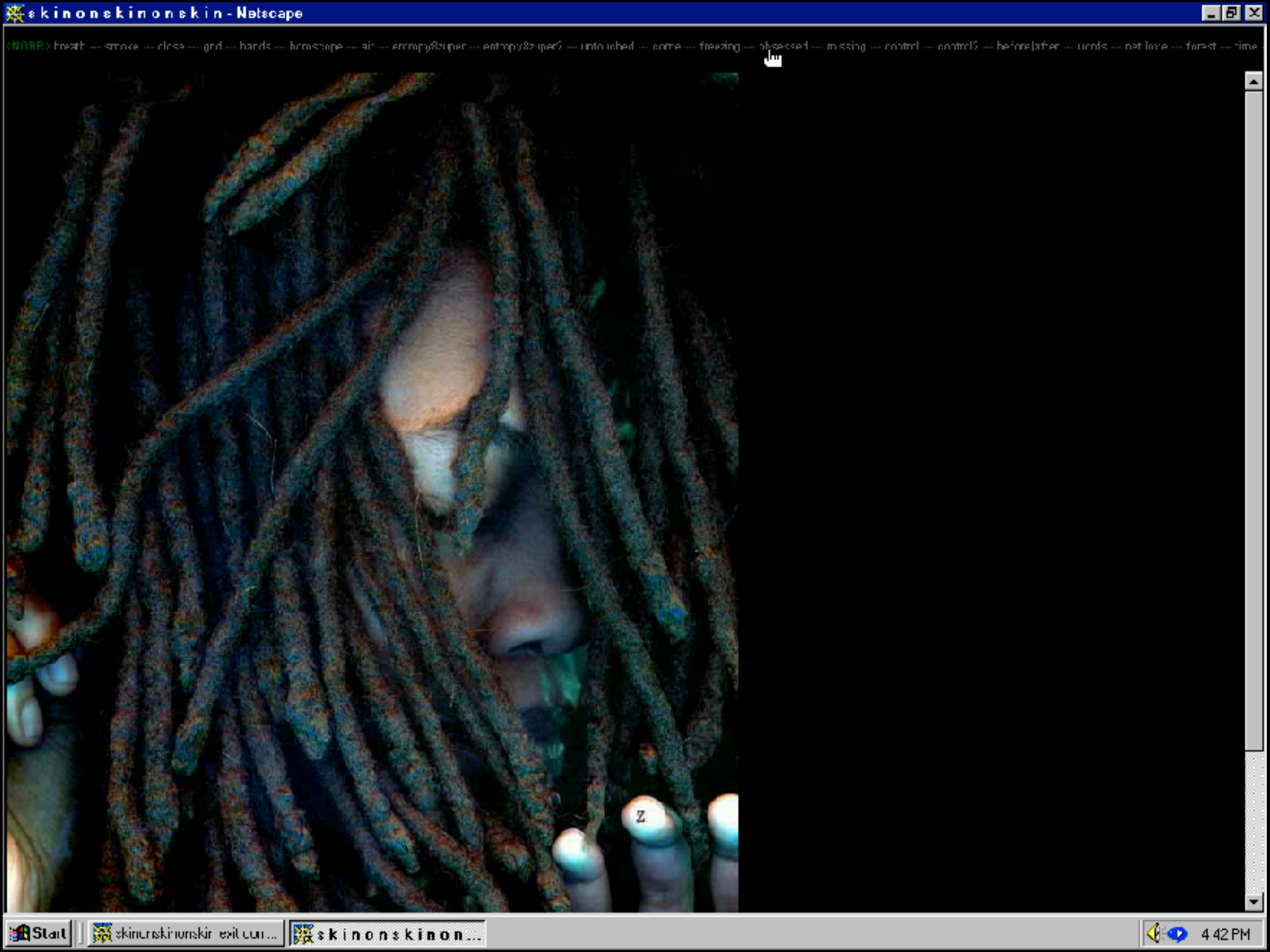
Brandon, 1998-1999 (2017)
Shu Lea Cheang
Restored in April 2017 by Emma Dickson



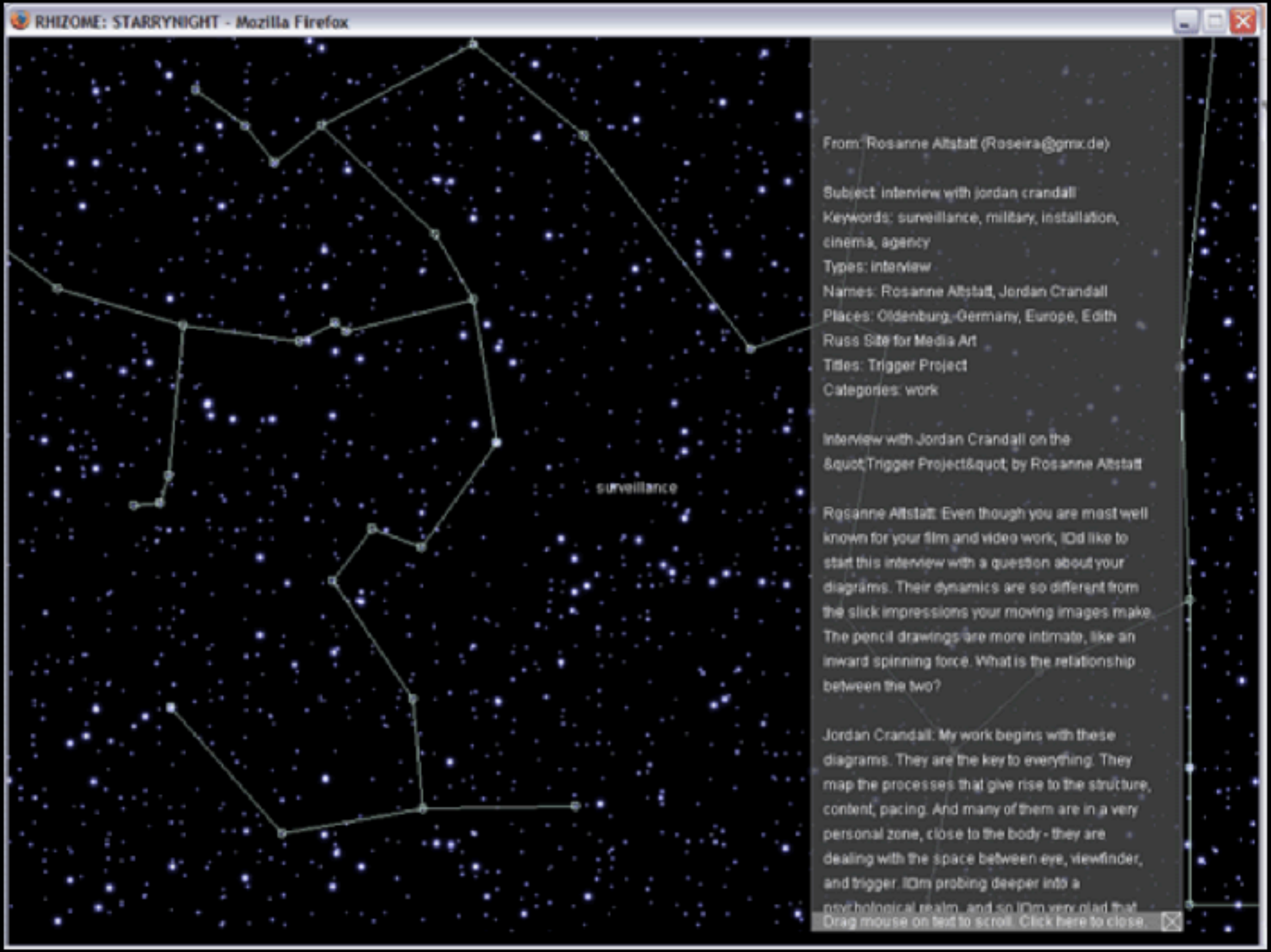
Click here for video tutorial of resored version, 1998-1999 (restored 2016-17)
Shu Lea Cheang



Brandon, 1998-199
Shu Lea Cheang


skinonskinonskin - Netscape
<NOBR> breath --- smoke --- close --- grid --- hands --- homescape --- air --- entropy8zuper --- entropy8zuper2 --- untouched --- come --- freezing --- obsessed --- missing --- control --- control2 --- before/after --- words --- net love --- forest --- time
Start
skinonskinonskin exit com...
skinonskinon...
4:42 PM

skinonskinonskin, 1999
Auriea Harvey (Entropy8) + Michaël Samyn (Zuper!)
Entropy8Zuper!



From: Rosanne Altstatt (Roseira@gmx.de)

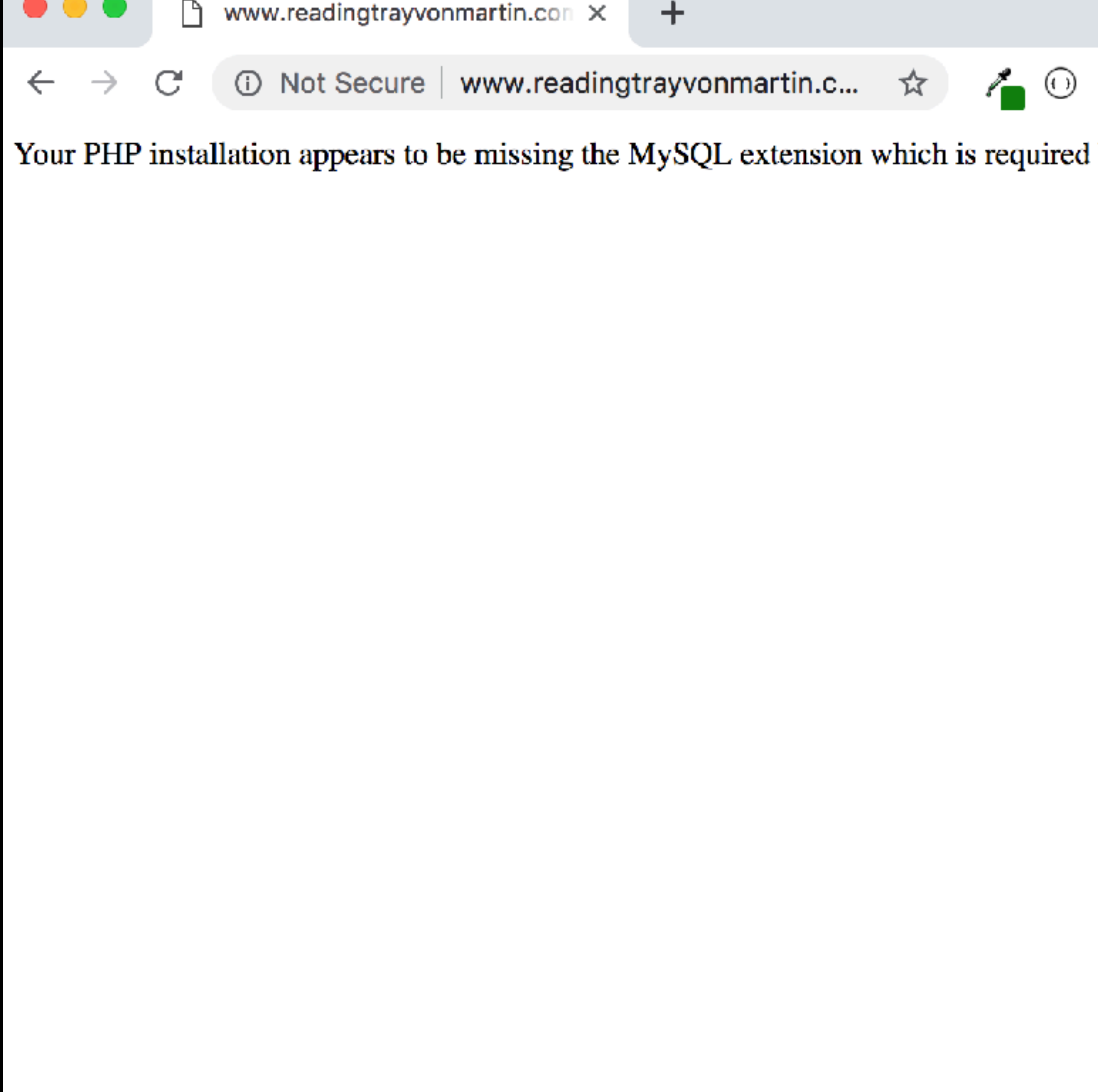
Subject: interview with jordan crandall
Keywords: surveillance, military, installation, cinema, agency
Types: interview
Names: Rosanne Altstatt, Jordan Crandall
Places: Oldenburg, Germany, Europe, Edith Russ Site for Media Art
Titles: Trigger Project
Categories: work

Interview with Jordan Crandall on the &quot;Trigger Project&quot; by Rosanne Altstatt

Rosanne Altstatt: Even though you are most well known for your film and video work, I0d like to start this interview with a question about your diagrams. Their dynamics are so different from the slick impressions your moving images make. The pencil drawings are more intimate, like an inward spinning force. What is the relationship between the two?

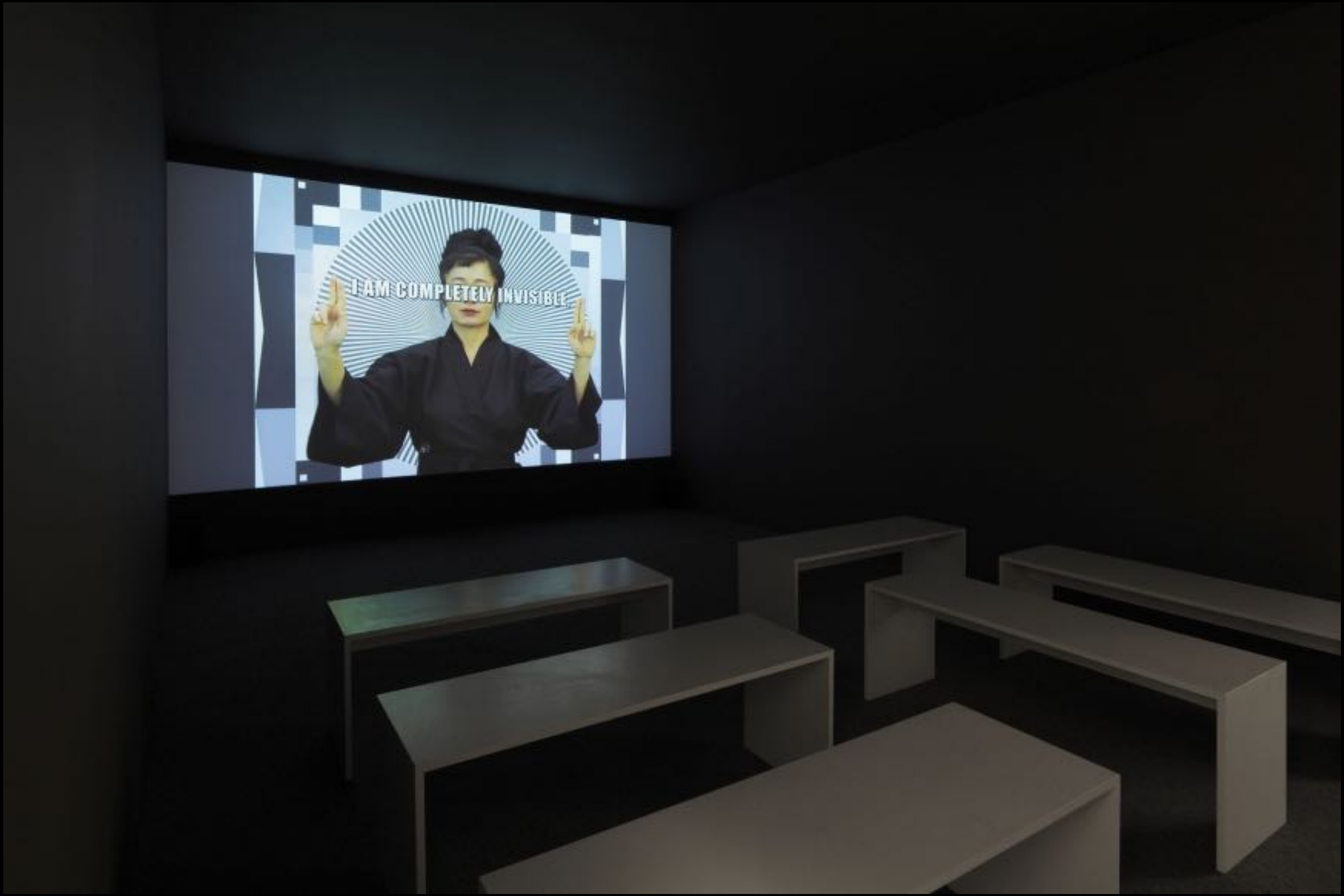
Jordan Crandall: My work begins with these diagrams. They are the key to everything. They map the processes that give rise to the structure, content, pacing. And many of them are in a very personal zone, close to the body - they are dealing with the space between eye, viewfinder, and trigger. I0m probing deeper into a psychological realm, and so I0m very glad that

Starry Night, 1999
Mark Tribe, Alexander Galloway, and Martin Wattenberg
@ Rhizome



Your PHP installation appears to be missing the MySQL extension which is required

Reading Trayvon Martin
Martine Sym



How Not to Be Seen: A Fucking Didatic Educational .MOV File

Video (color, sound), 14 min

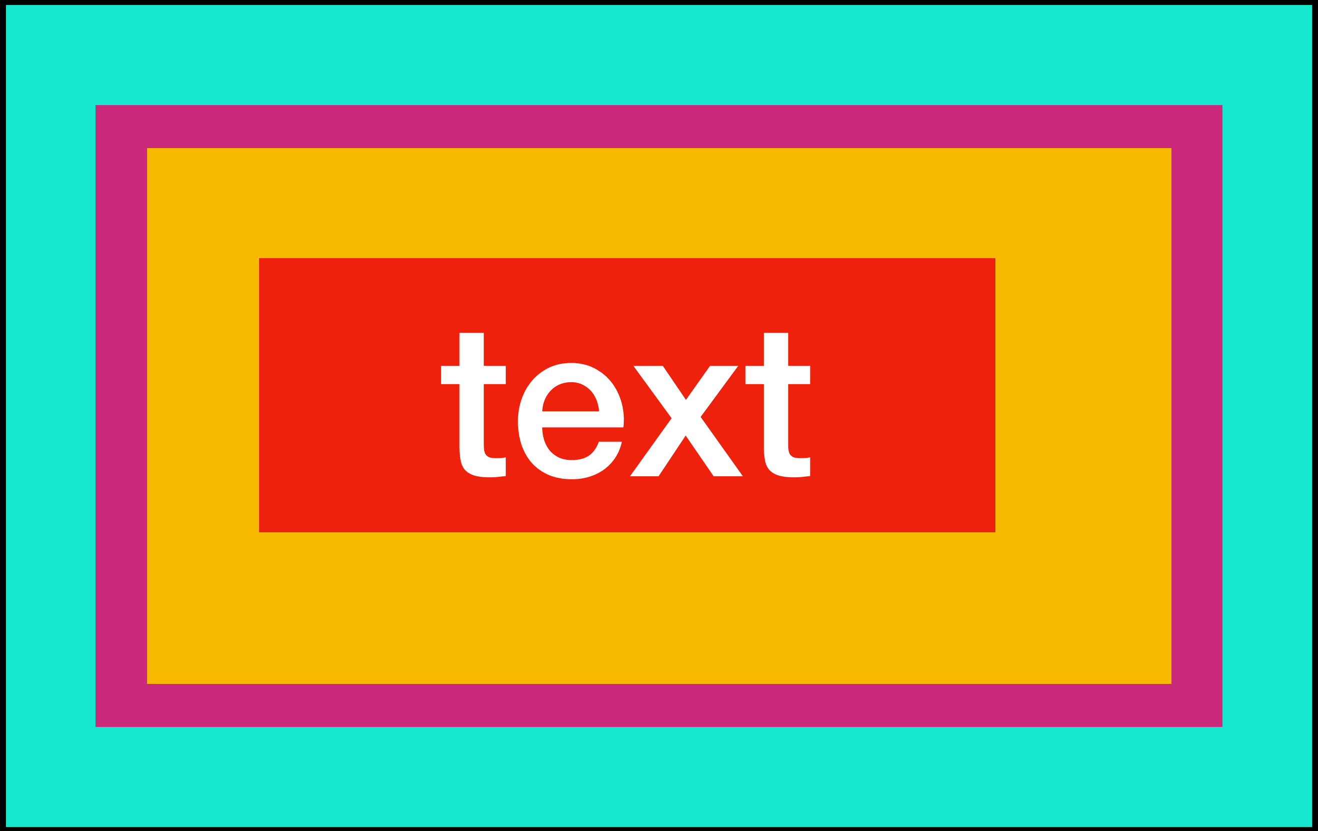
Hito Steyerl, 2013

Image: How Not to Be Seen: A Fucking Didactic Educational Installation, 2014 review

Border

All boxes have borders even if invisible or 0px wide. It separates the edge of one box from another.

Box Model



Padding

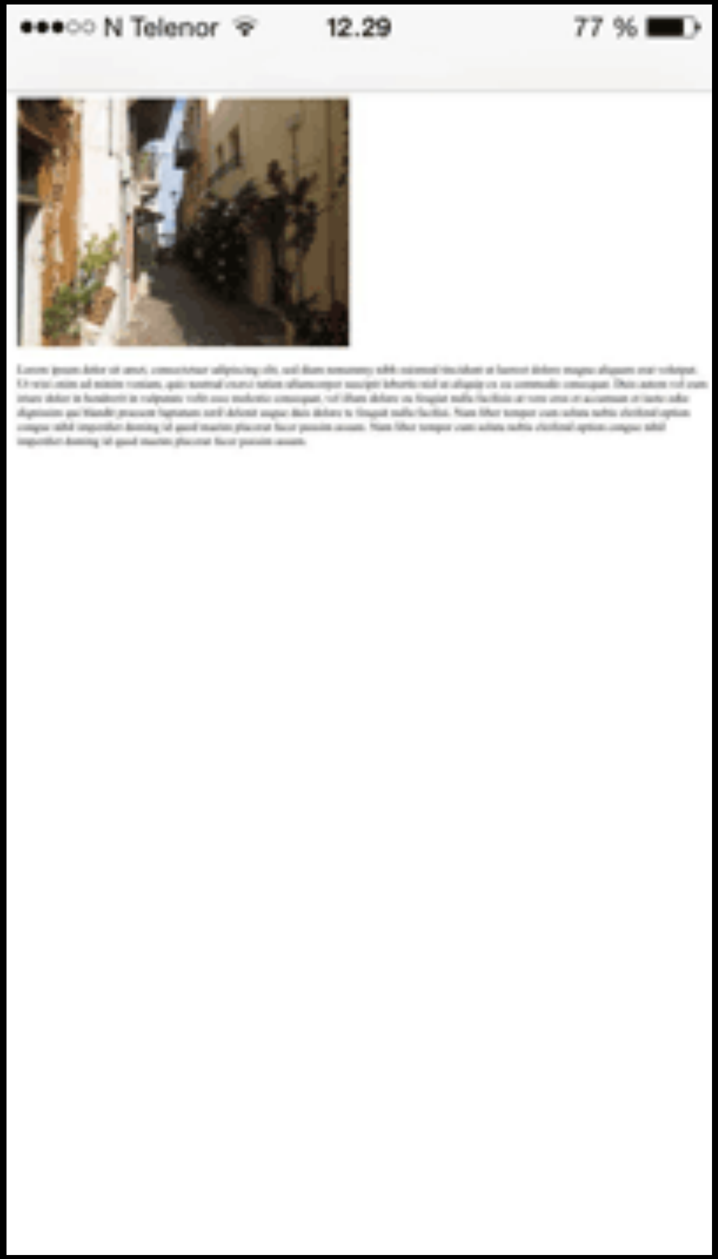
Padding is the space btw the border + any content contained within it. More padding increases the readability of its contents.

Margin

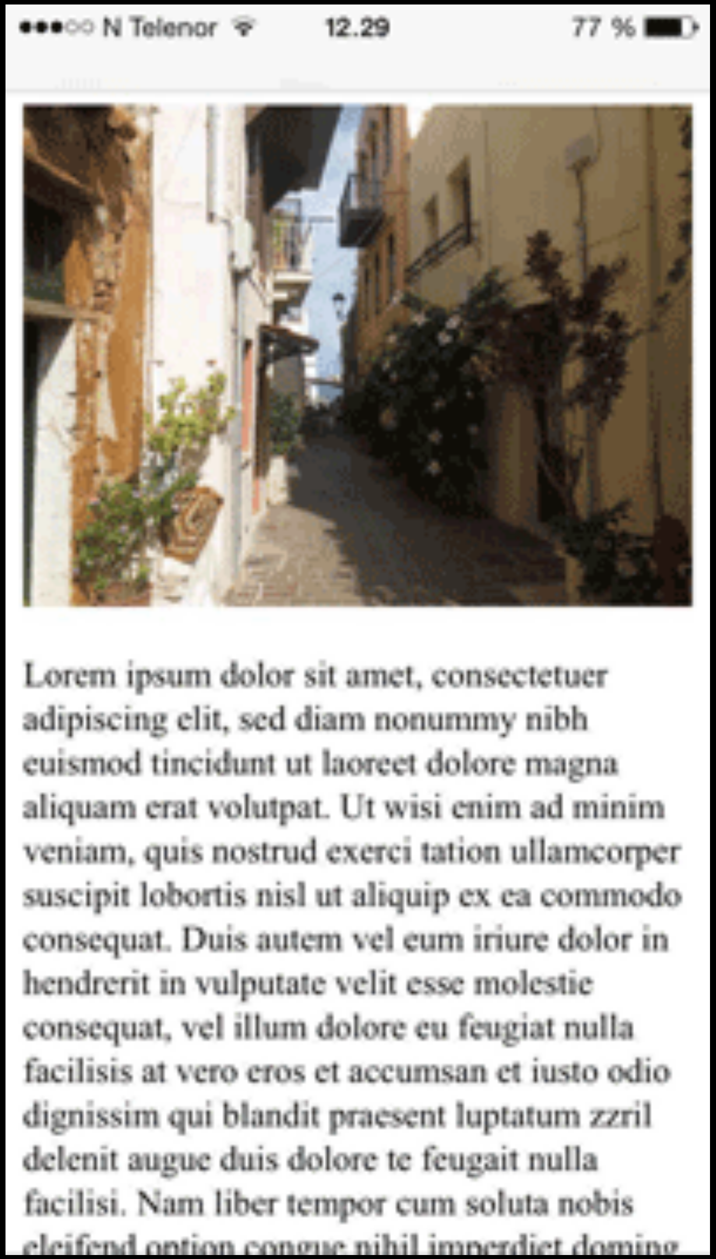
Content

Margins sit outside the edge of the border. You can set the width to create a gap btw borders of adjacent boxes.

```
<meta name="viewport" content="width=device-width, initial-scale=1.0”>
```



without



Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming

with

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

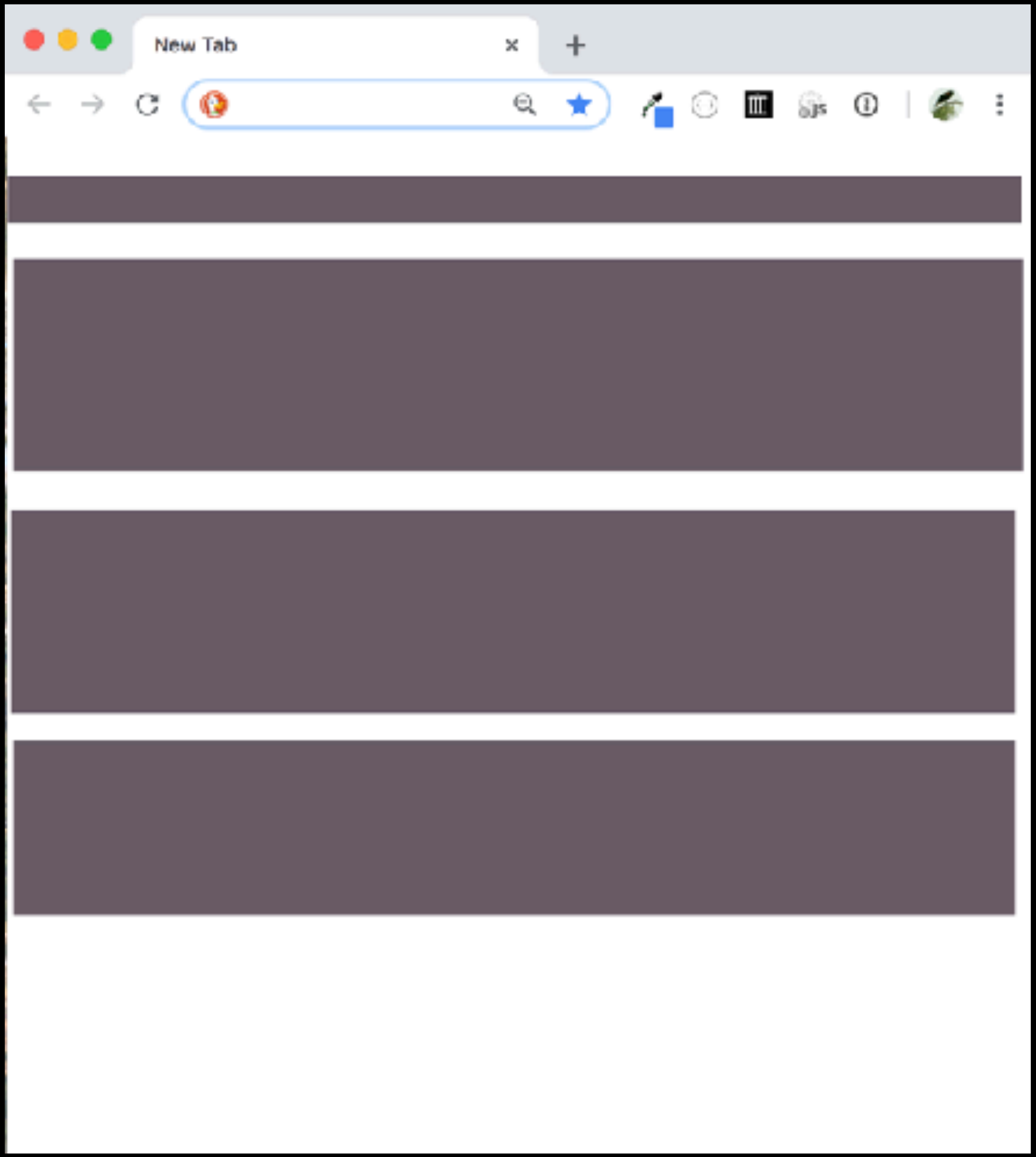
- 1vh = 1/100th of the viewport height

- 1vw = 1/100th of the viewport width

Example:

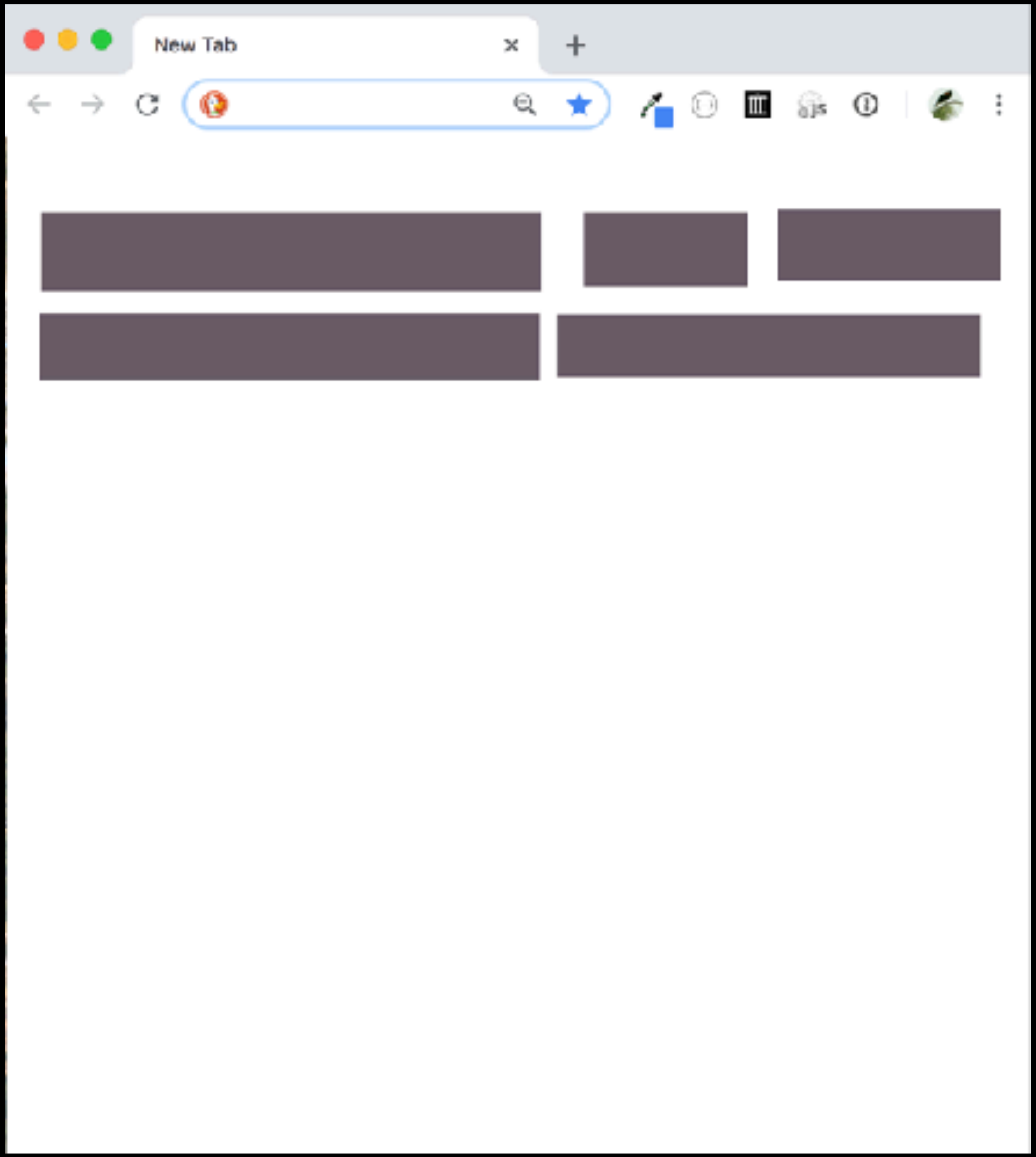
- height: 100vh; - width: 100vw;

flexbox



Block layout

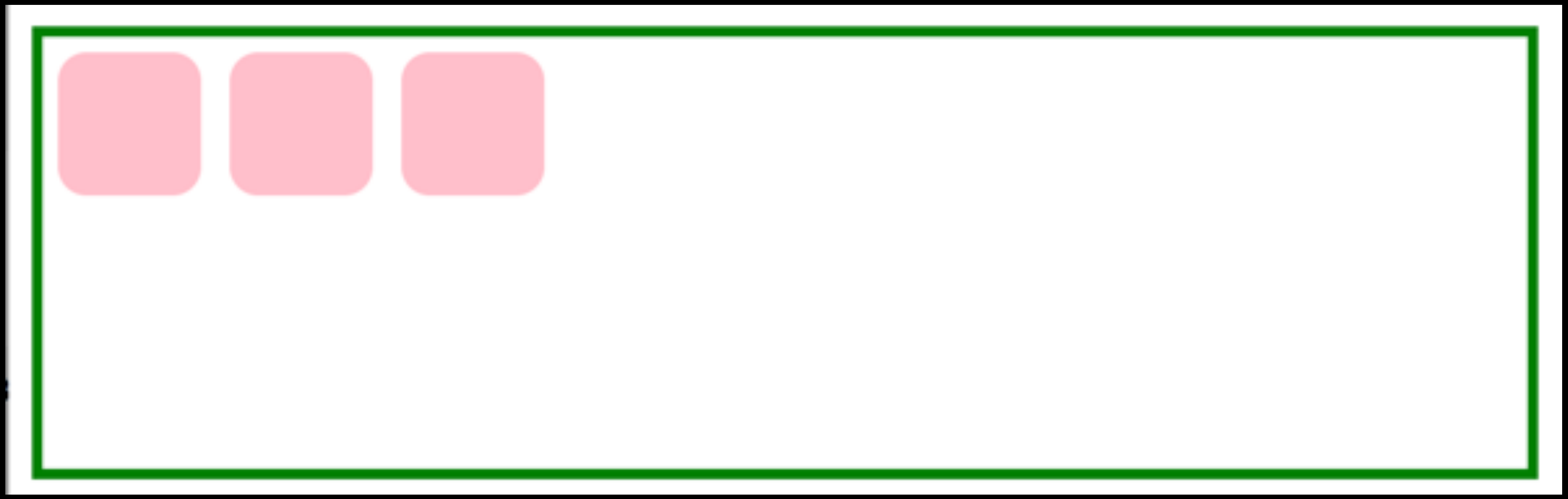
Laying out large sections of a page

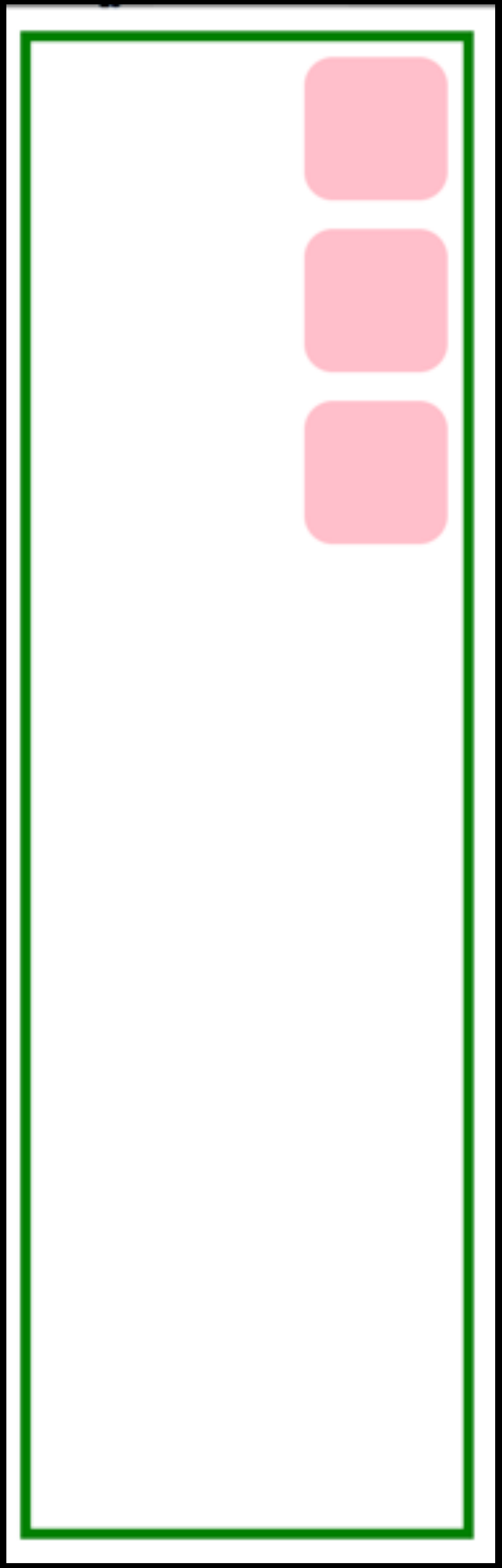


Inline layout

Laying out txt + other inline content within a section

Flex Layout



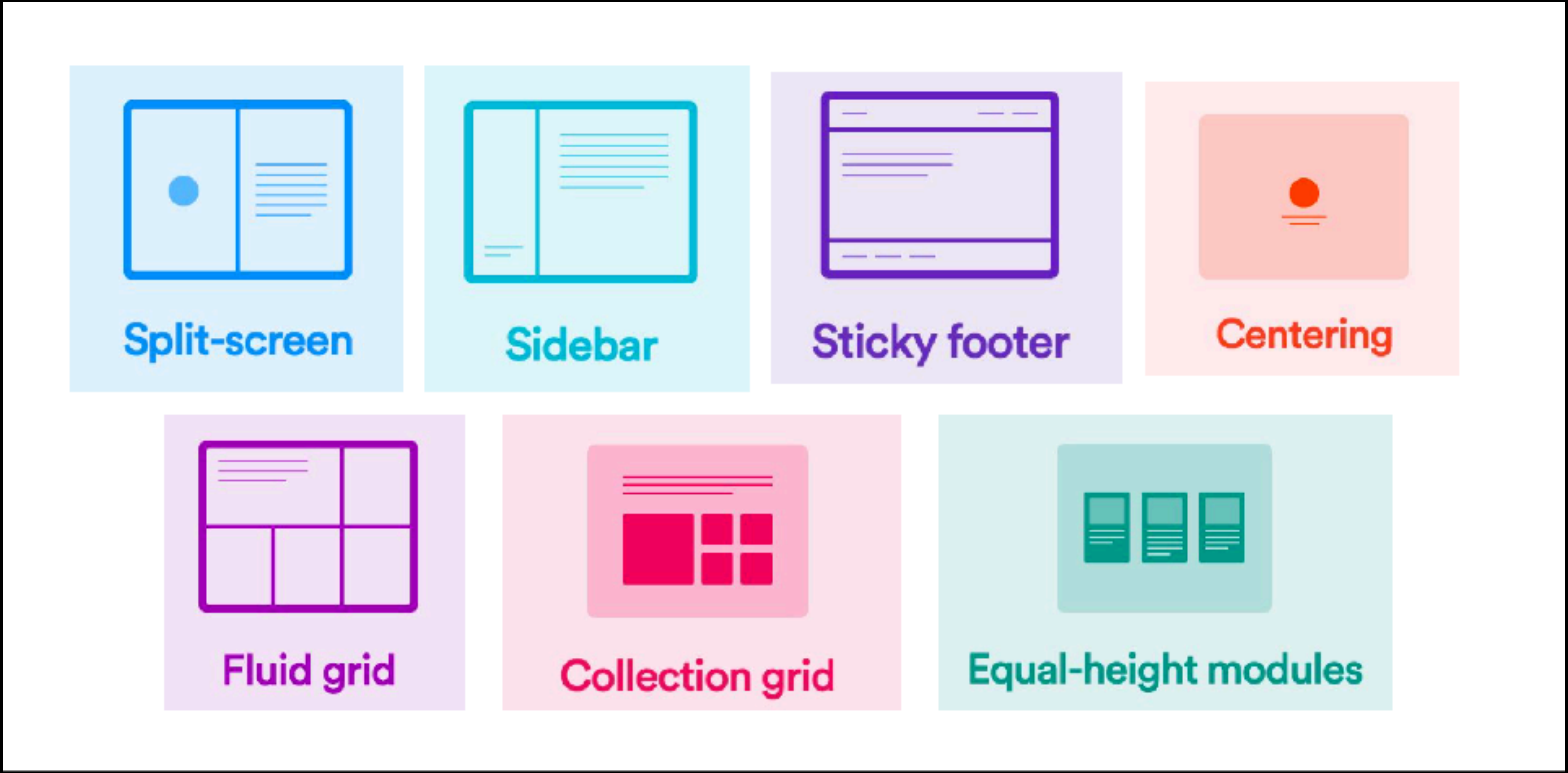


To achieve more complicated layouts, we can enable a different kind of CSS layout rendering mode: **Flex layout.**

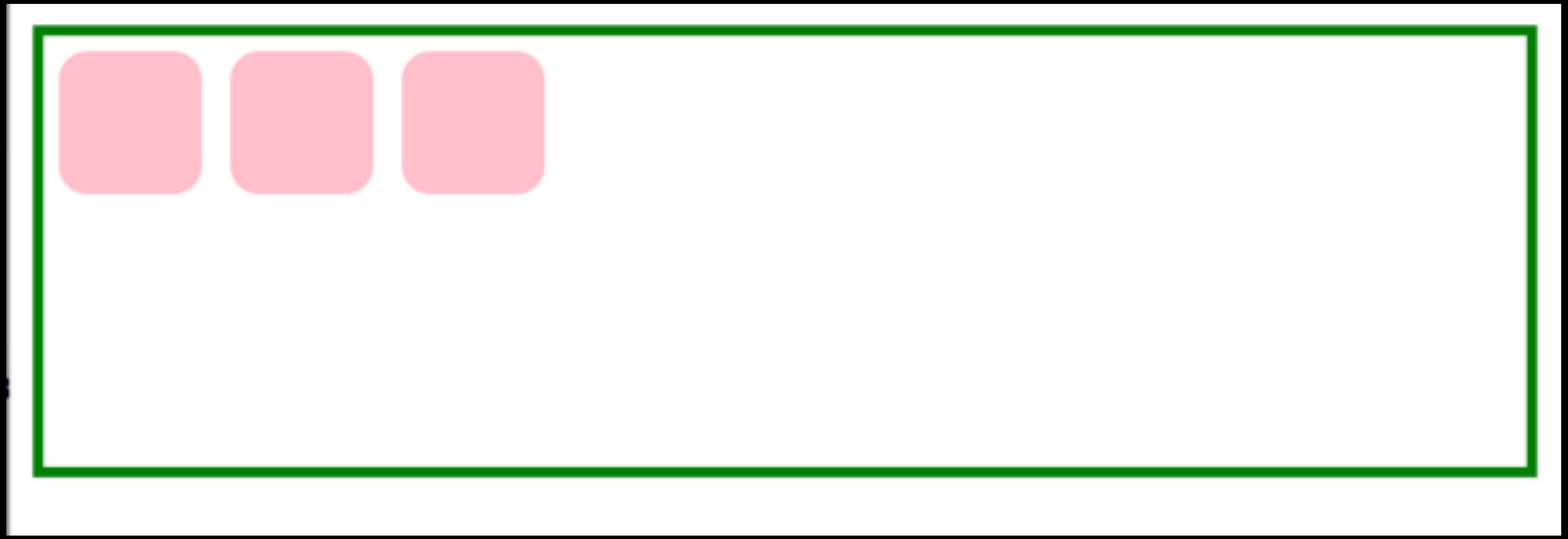
Flex layout defines a special set of rules for laying out items in rows or columns.

Flex Layout

Here are examples of layouts that are easy w/ flex layout + really difficult otherwise.



Flex Basics



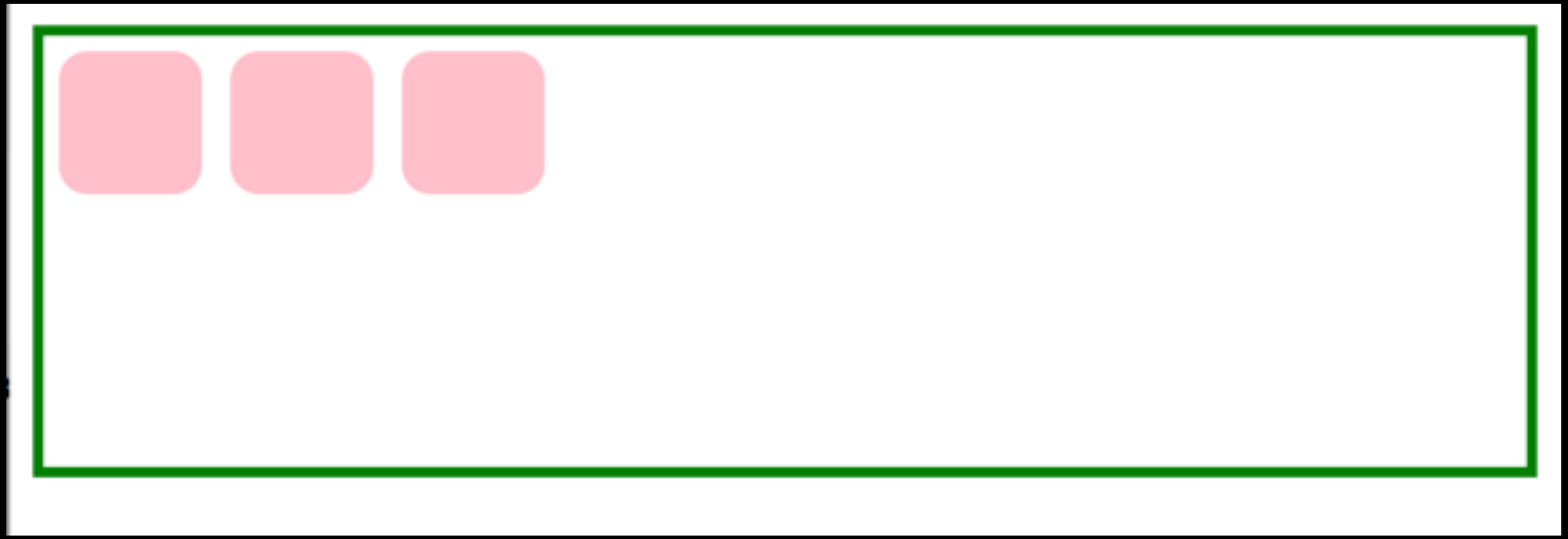
Flex layouts are composed of:

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 item(s)** are displayed

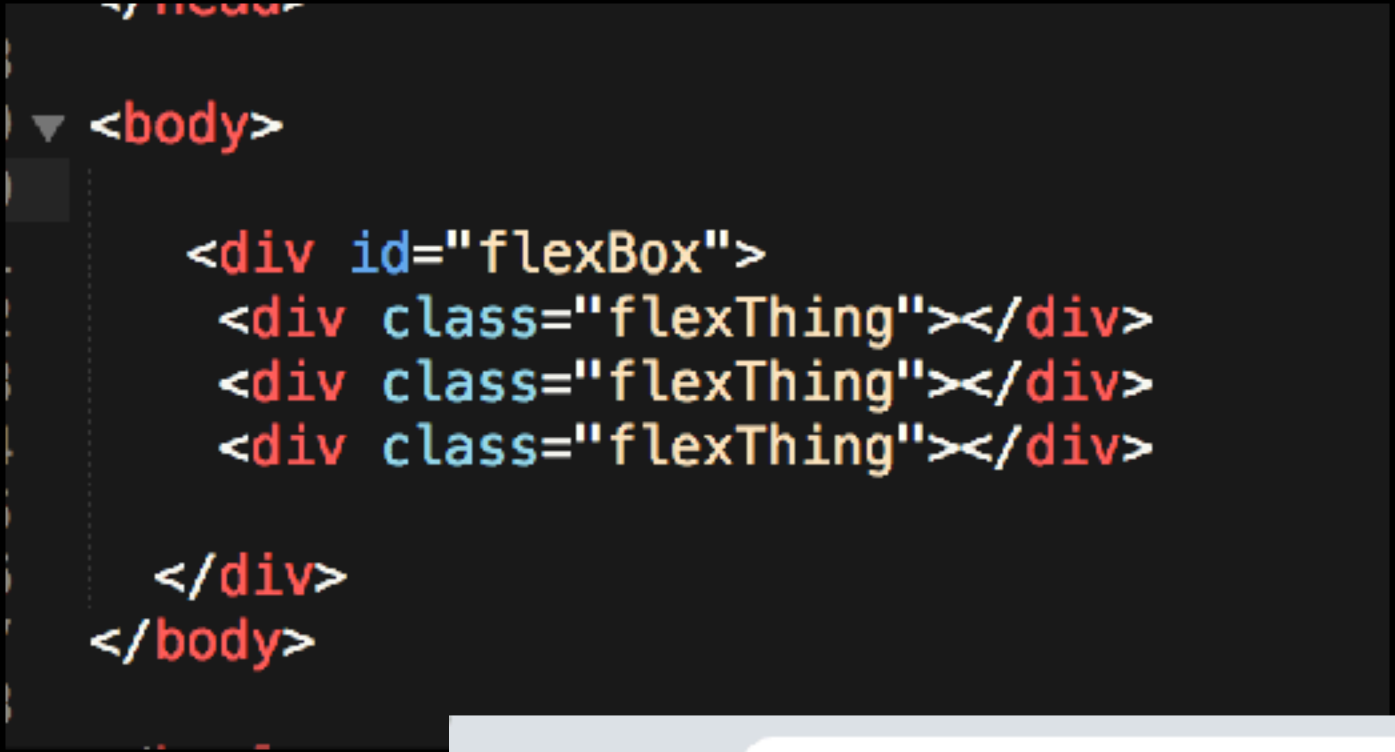
Flex Basics



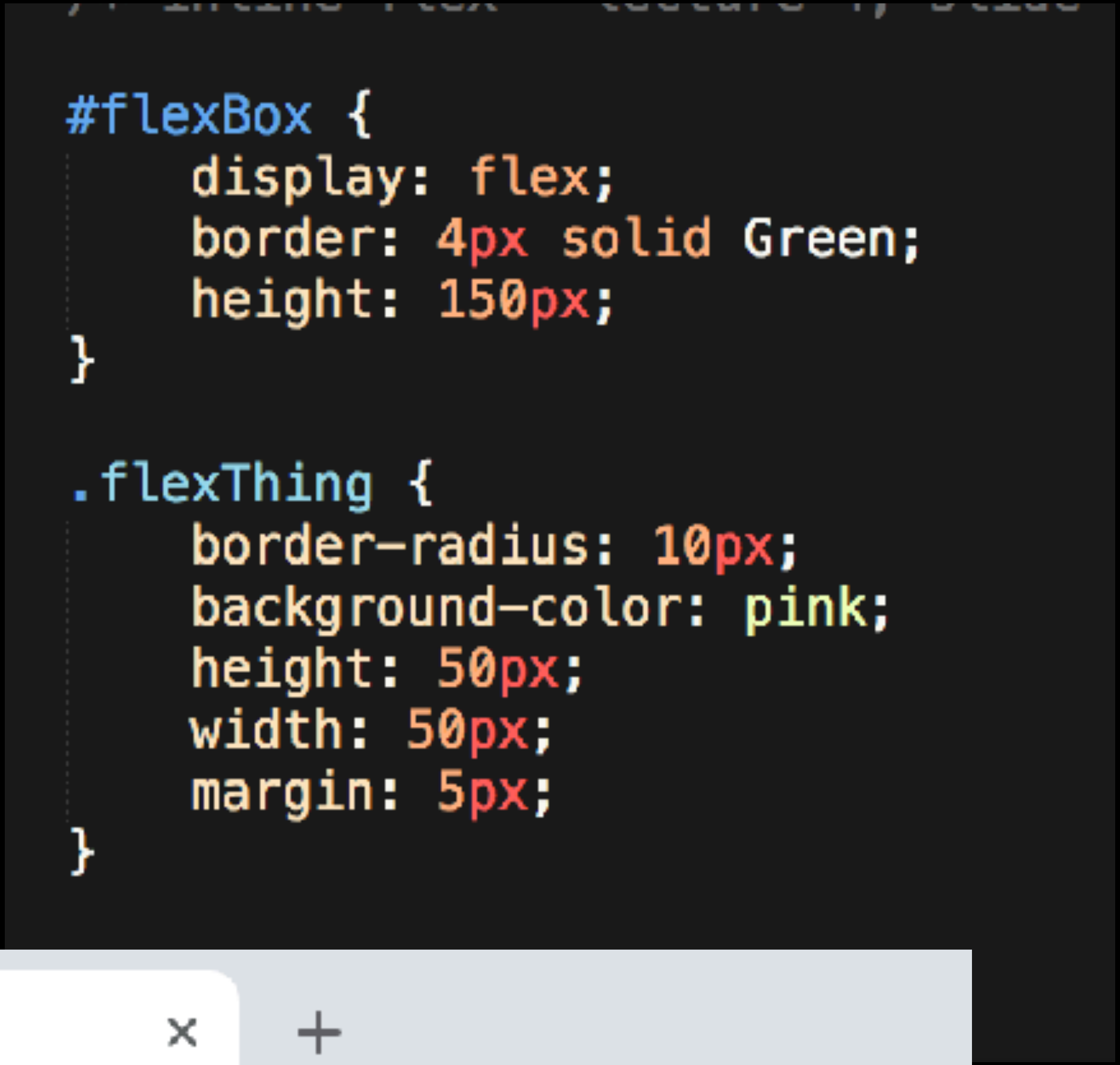
To make an element a flex container, change display:

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

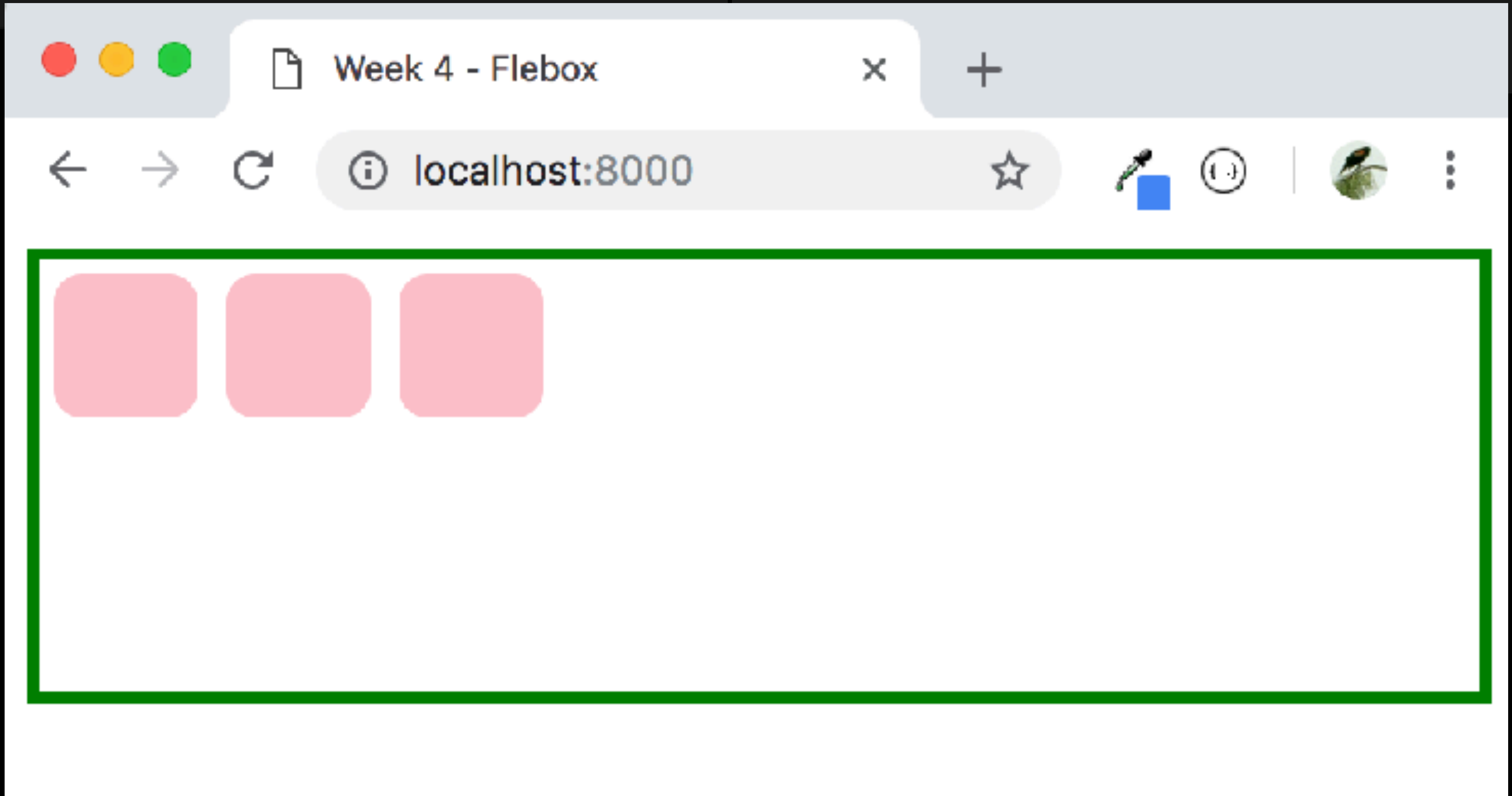
Flex Basics



```
<body>

   <div id="flexBox">
    <div class="flexThing"></div>
    <div class="flexThing"></div>
    <div class="flexThing"></div>

  </div>
</body>
```

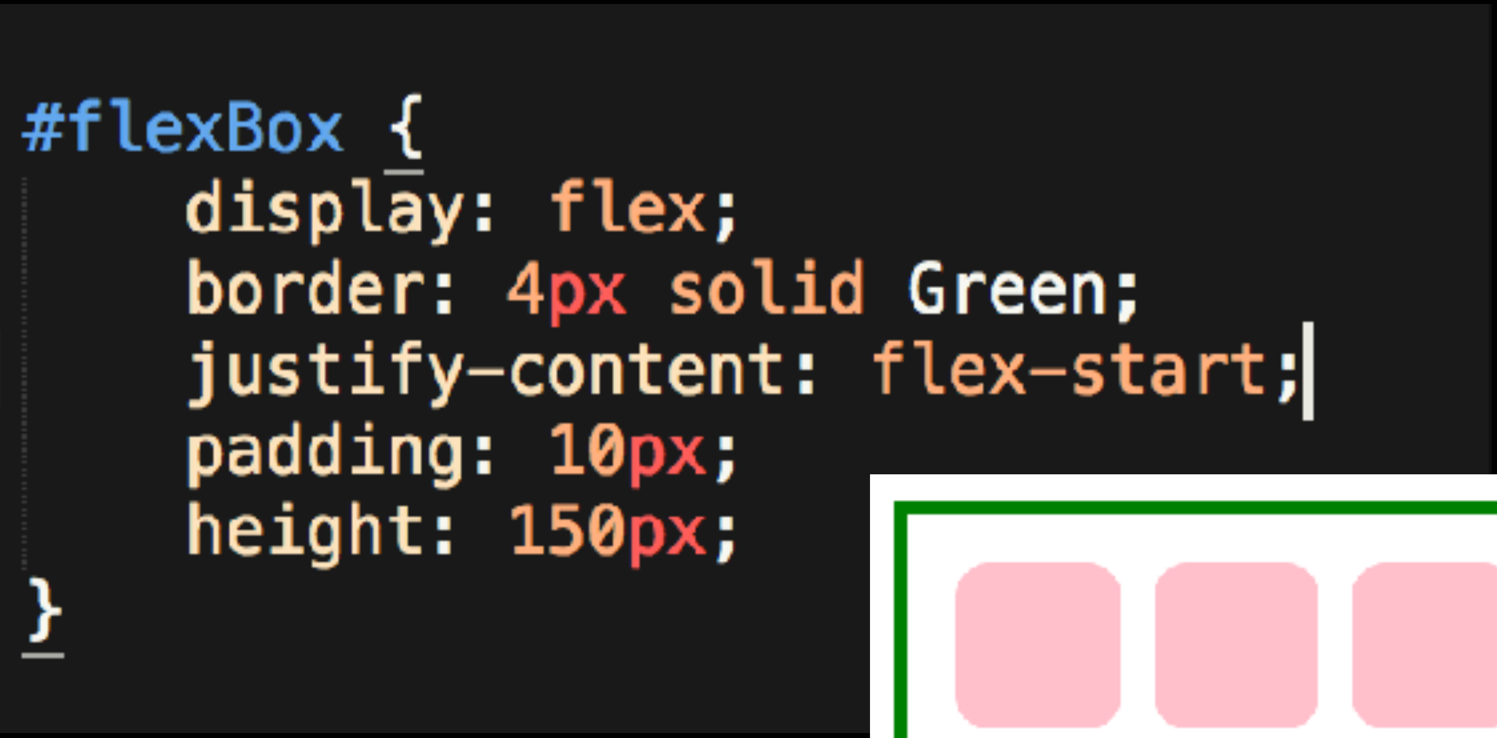


```
#flexBox {
    display: flex;
    border: 4px solid Green;
    height: 150px;
}

.flexThing {
    border-radius: 10px;
    background-color: pink;
    height: 50px;
    width: 50px;
    margin: 5px;
}
```

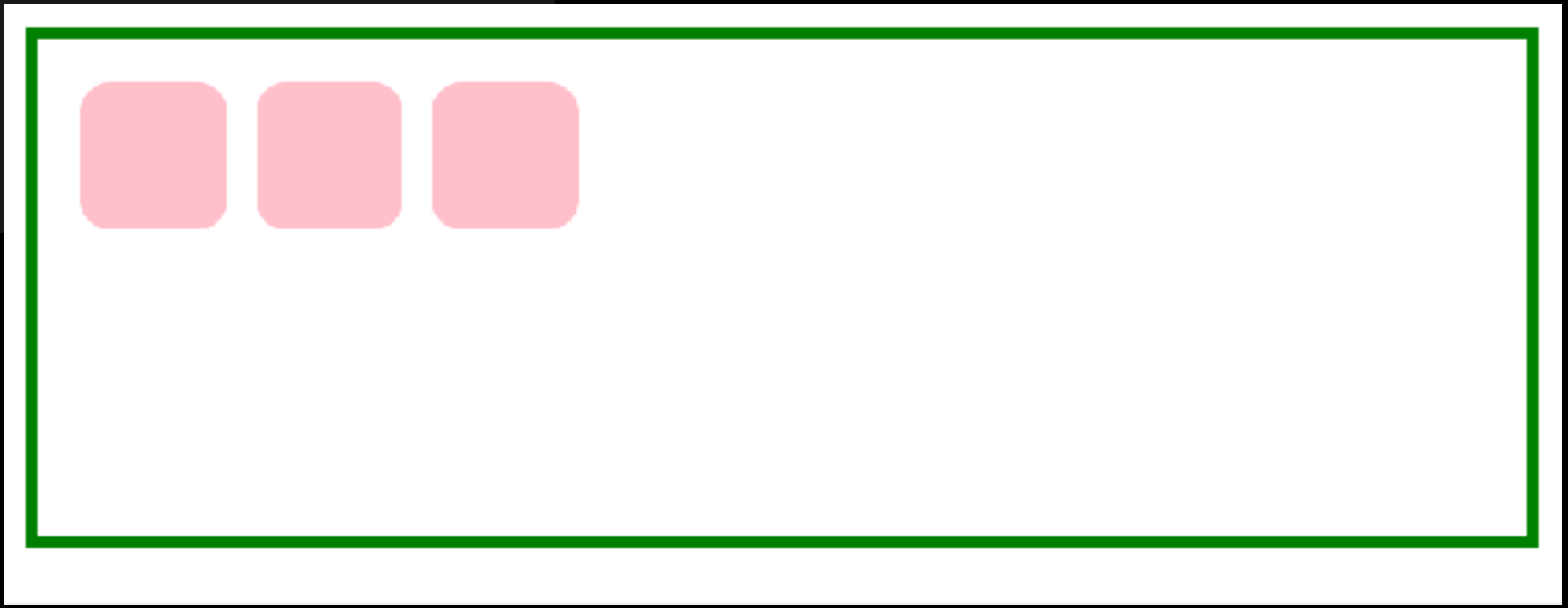


Flex Basics: justify-content

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

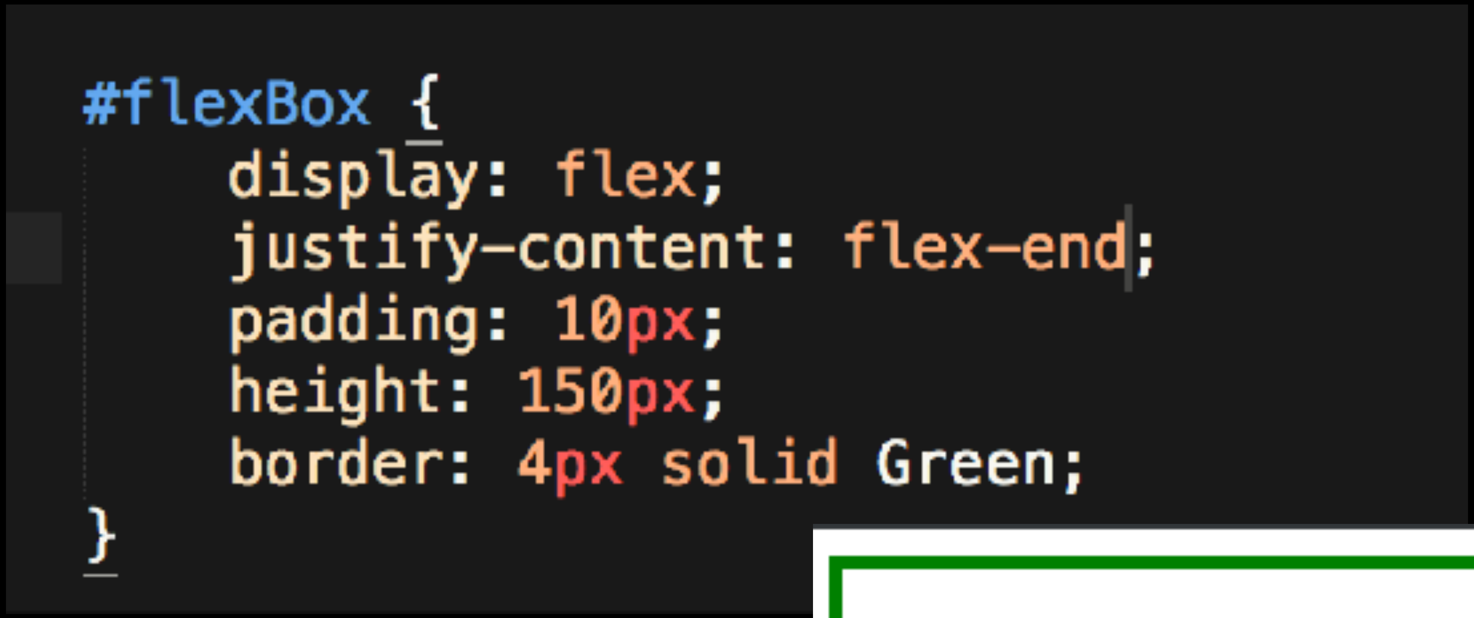


```
#flexBox {
    display: flex;
    border: 4px solid Green;
    justify-content: flex-start;
    padding: 10px;
    height: 150px;
}
```

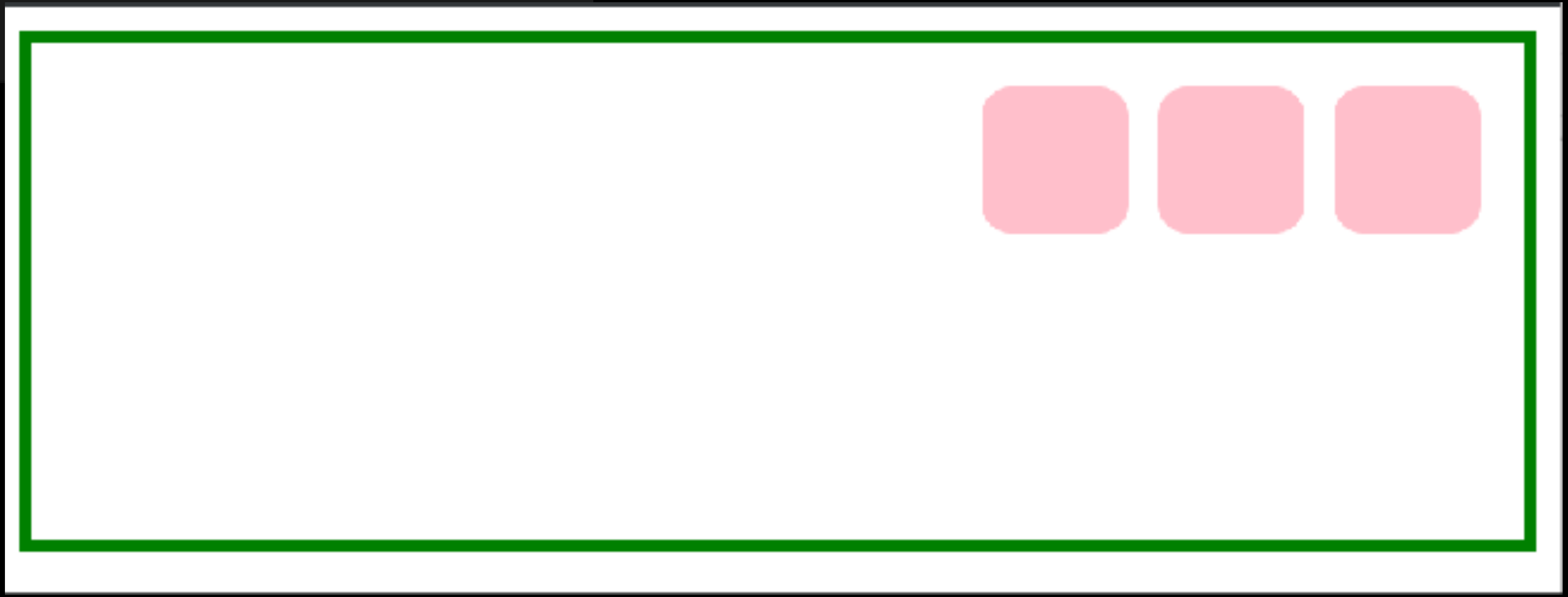


Flex Basics: justify-content

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

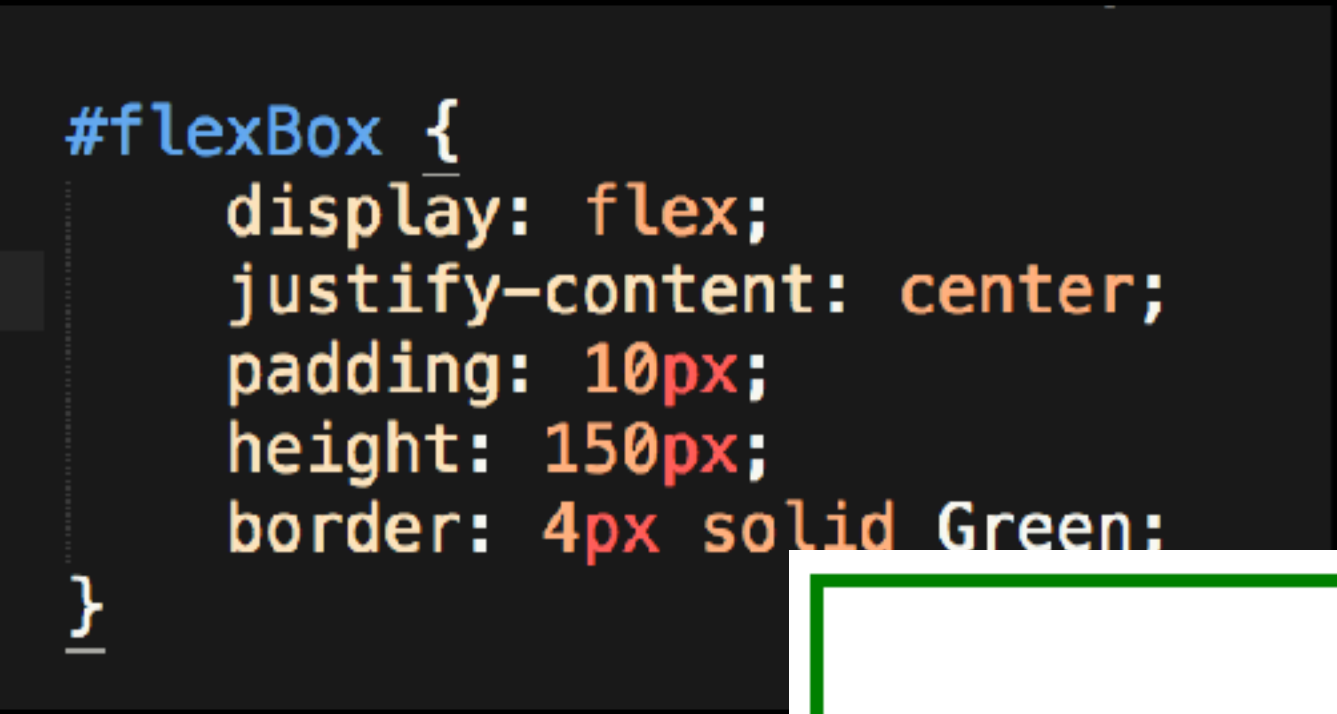


```
#flexBox {
    display: flex;
    justify-content: flex-end;
    padding: 10px;
    height: 150px;
    border: 4px solid Green;
}
```

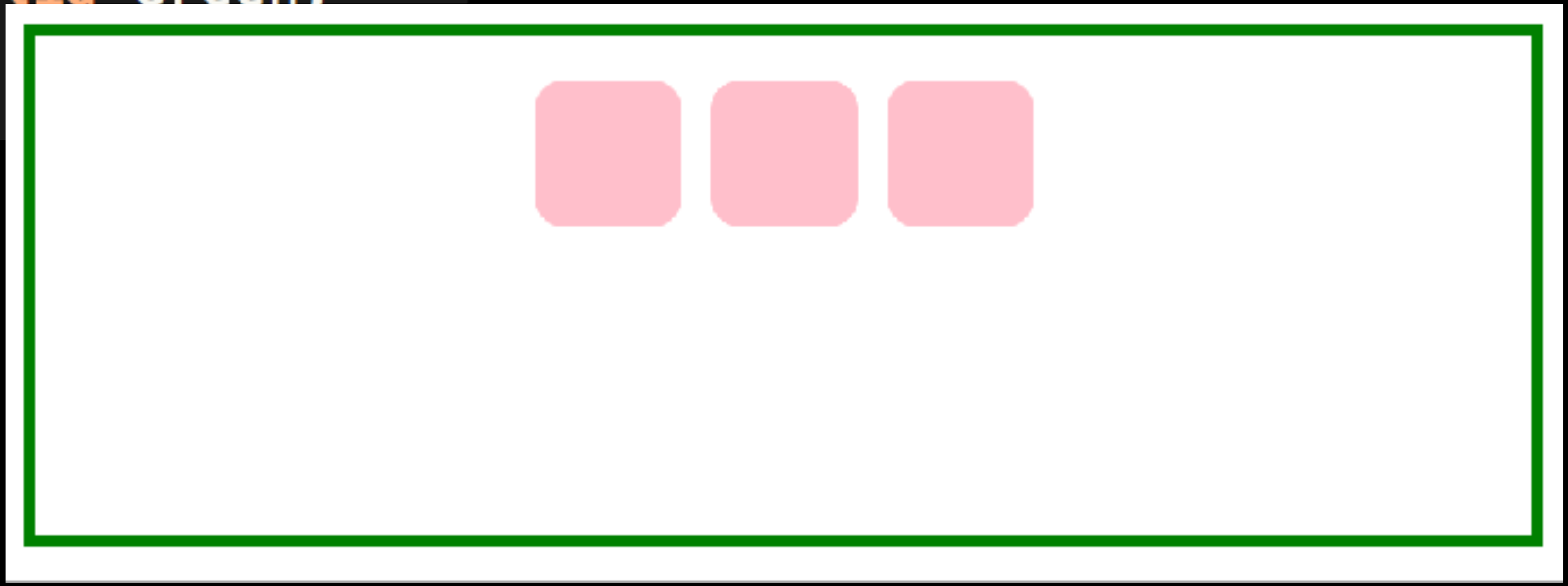


Flex Basics: justify-content

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

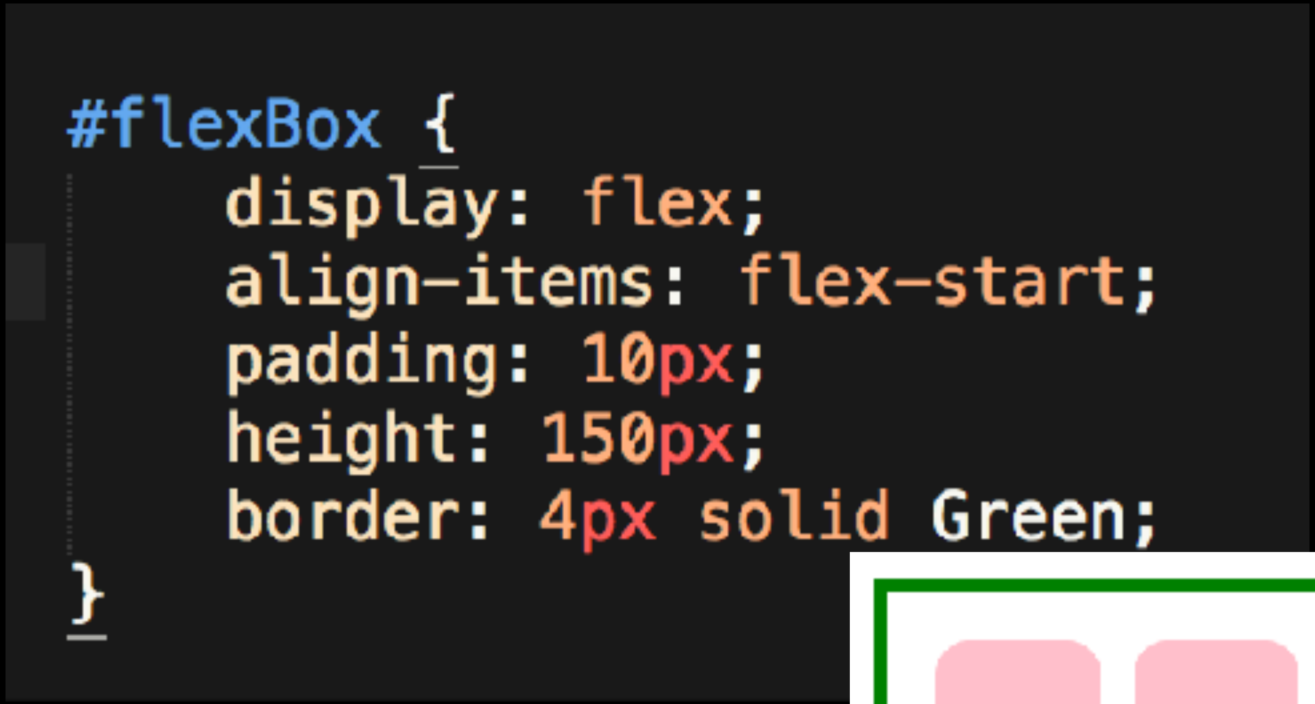


```
#flexBox {
    display: flex;
    justify-content: center;
    padding: 10px;
    height: 150px;
    border: 4px solid Green;
}
```

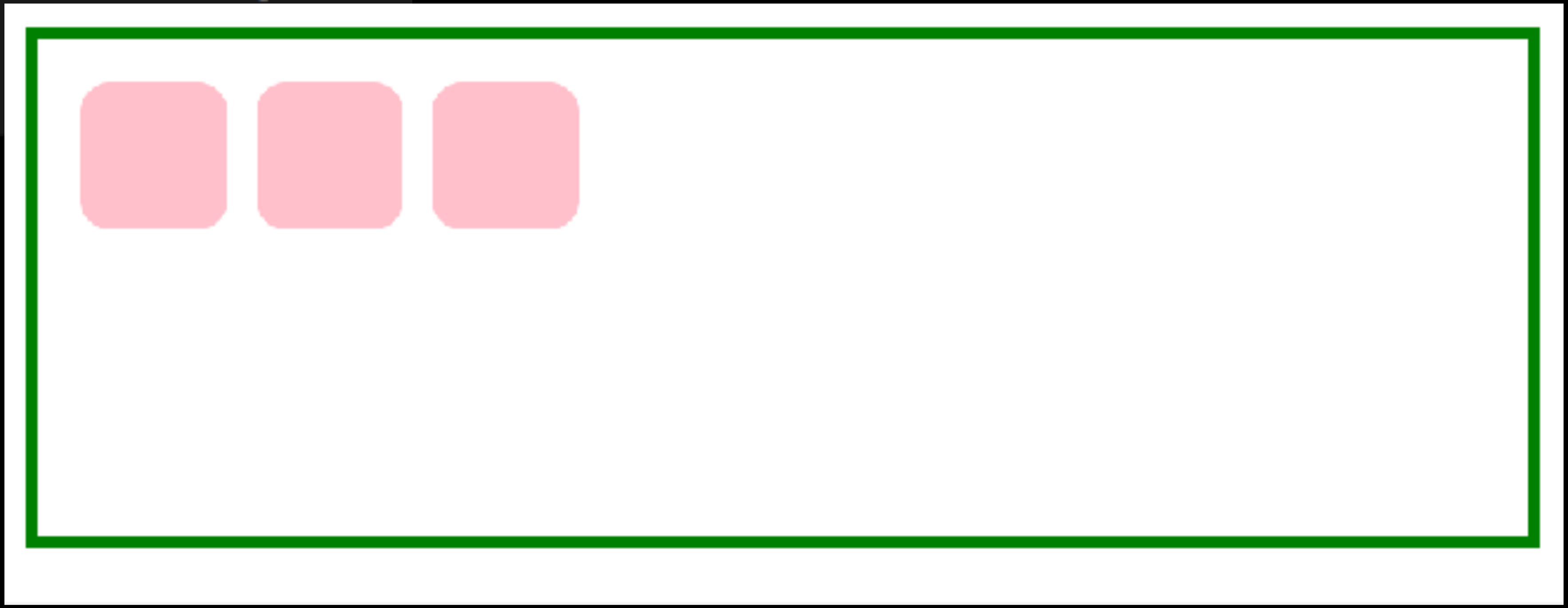


Flex Basics: align-items

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

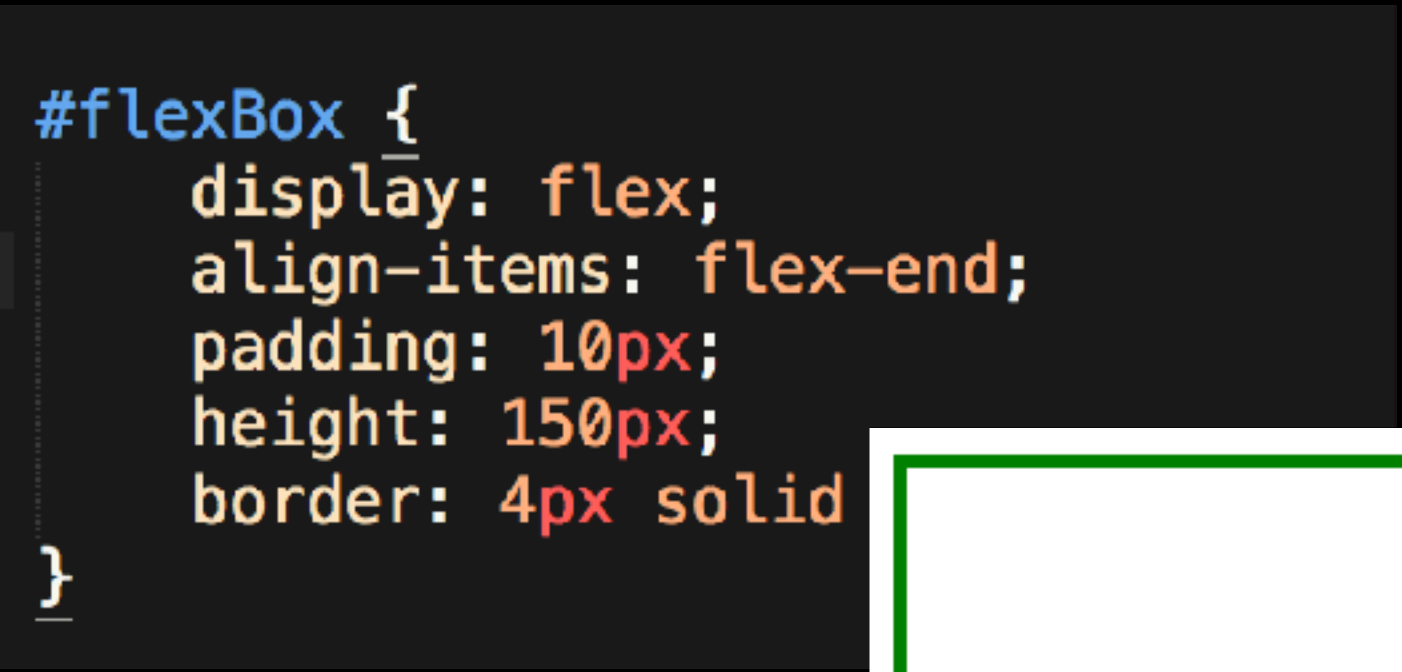


```css
#flexBox {
    display: flex;
    align-items: flex-start;
    padding: 10px;
    height: 150px;
    border: 4px solid Green;
}
```

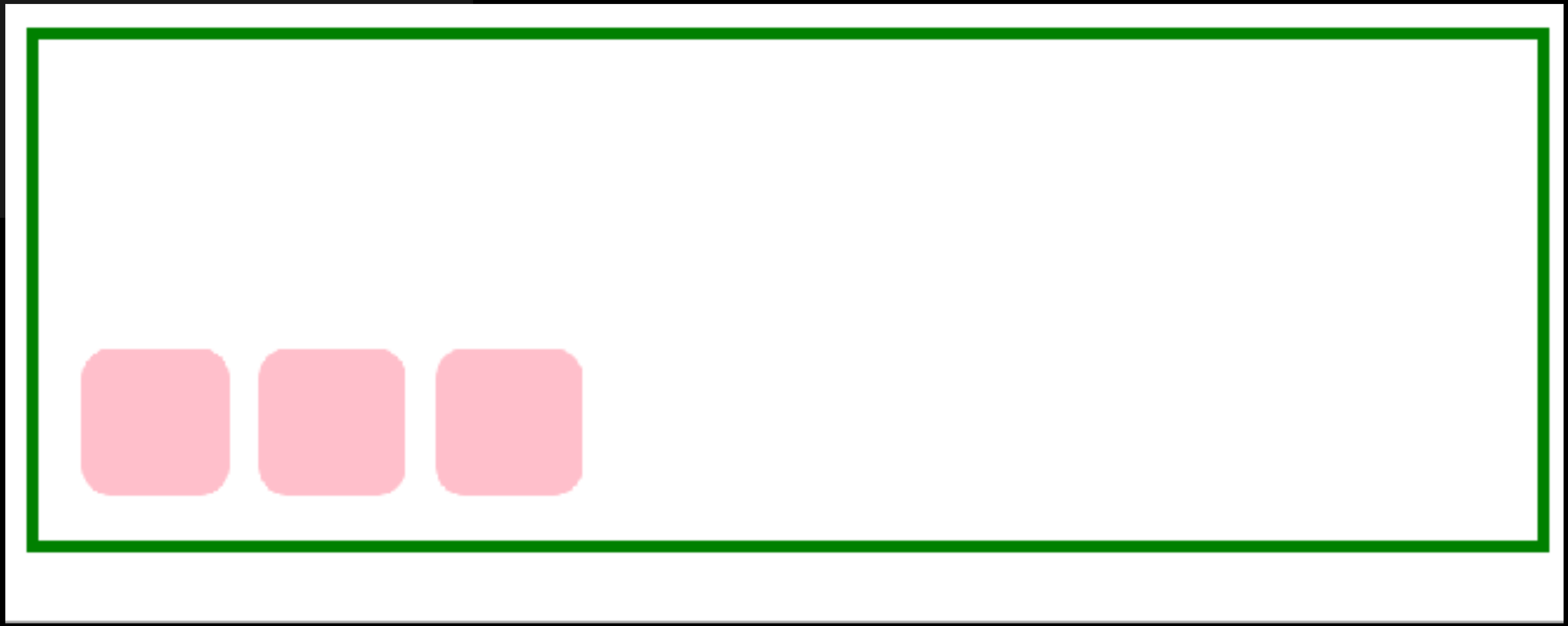


Flex Basics: align-items

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

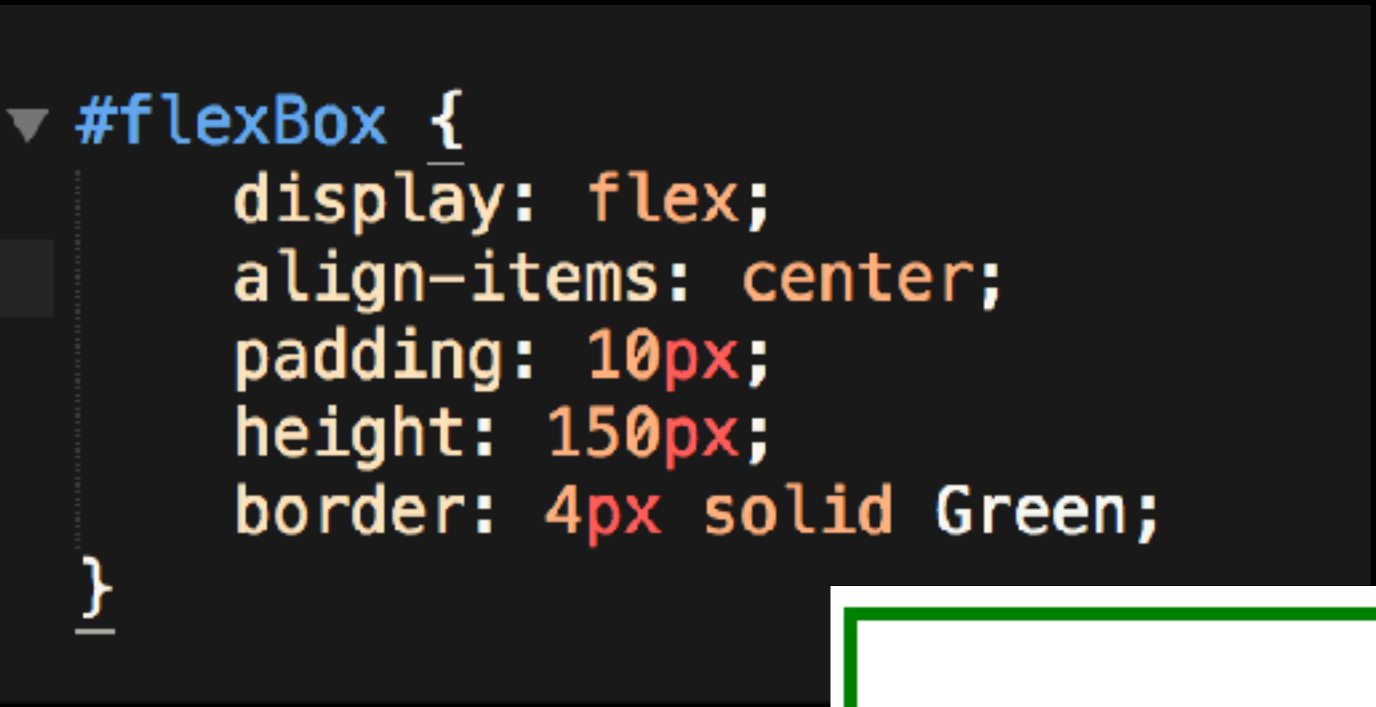


```
#flexBox {
    display: flex;
    align-items: flex-end;
    padding: 10px;
    height: 150px;
    border: 4px solid
}
```

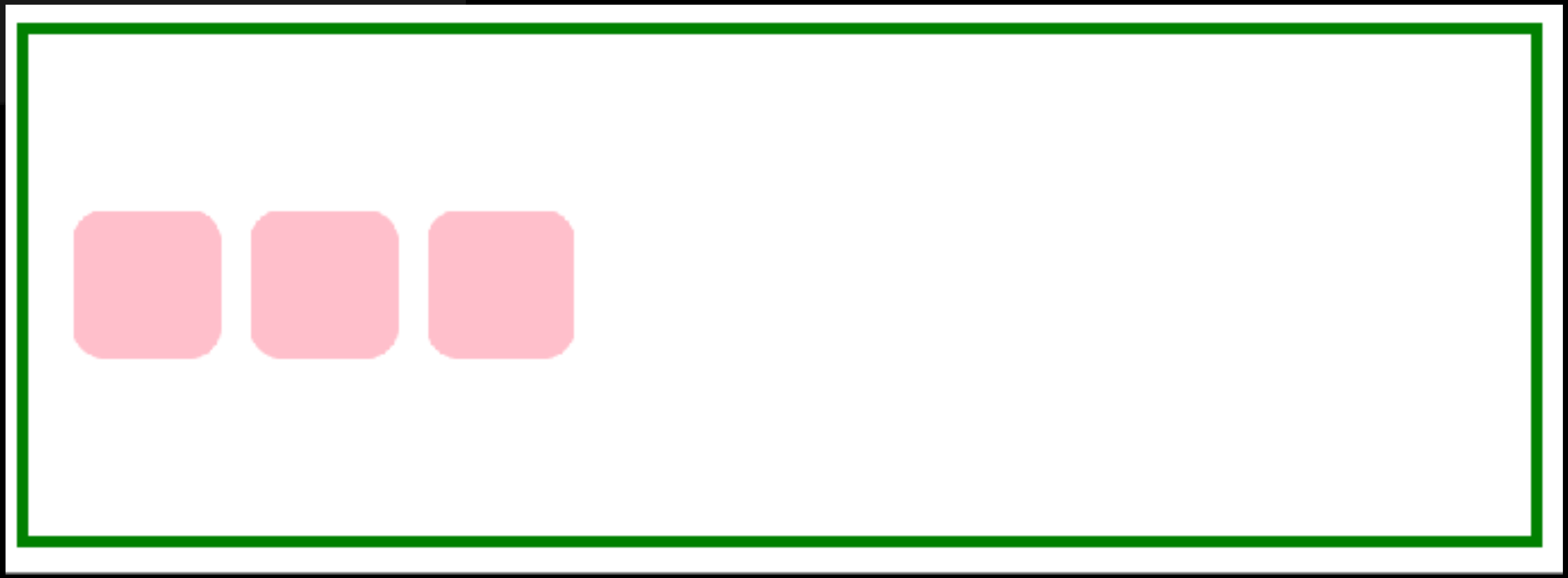


Flex Basics: align-items

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

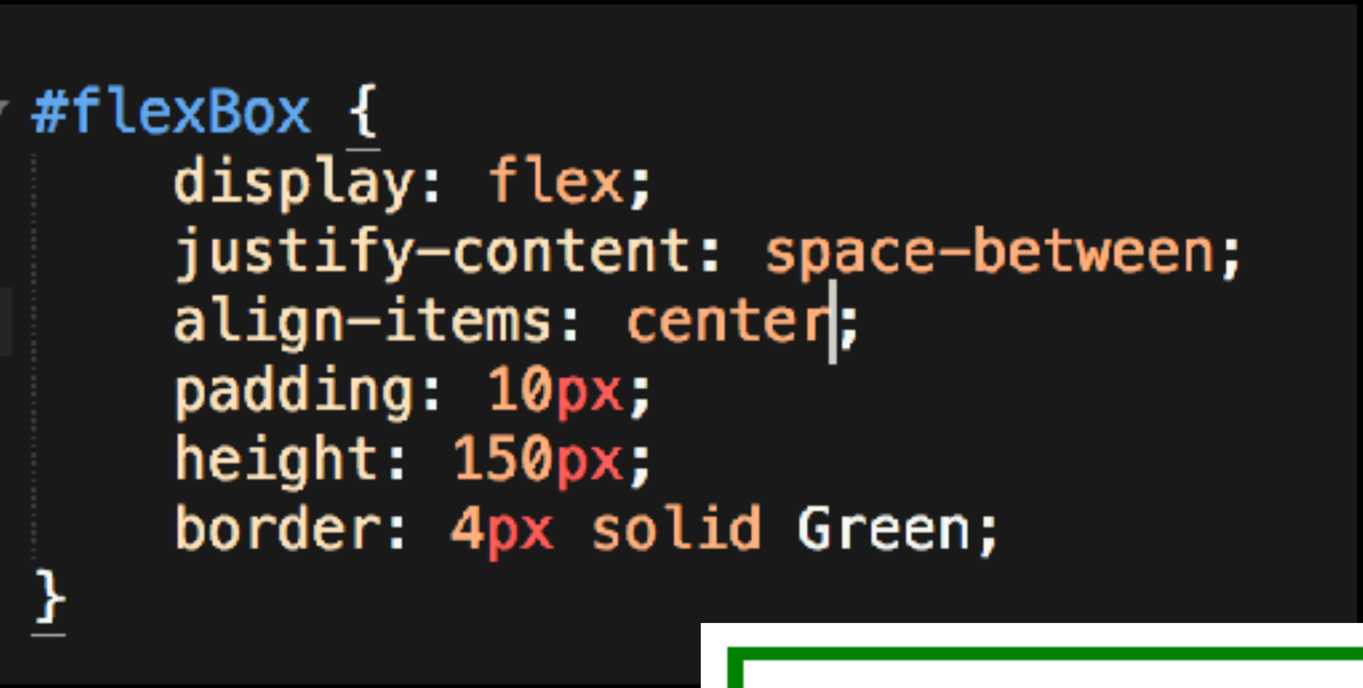


```
#flexBox {
    display: flex;
    align-items: center;
    padding: 10px;
    height: 150px;
    border: 4px solid Green;
}
```

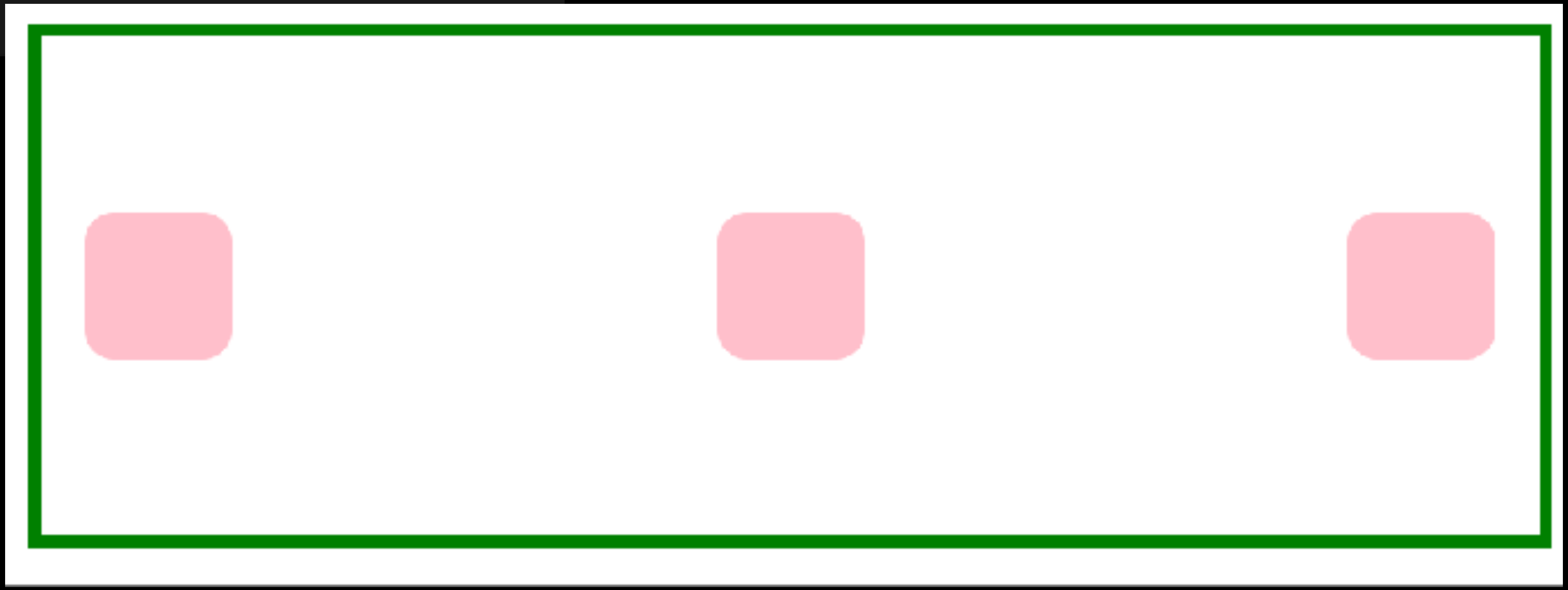


Flex Basics:

space-between + space-around

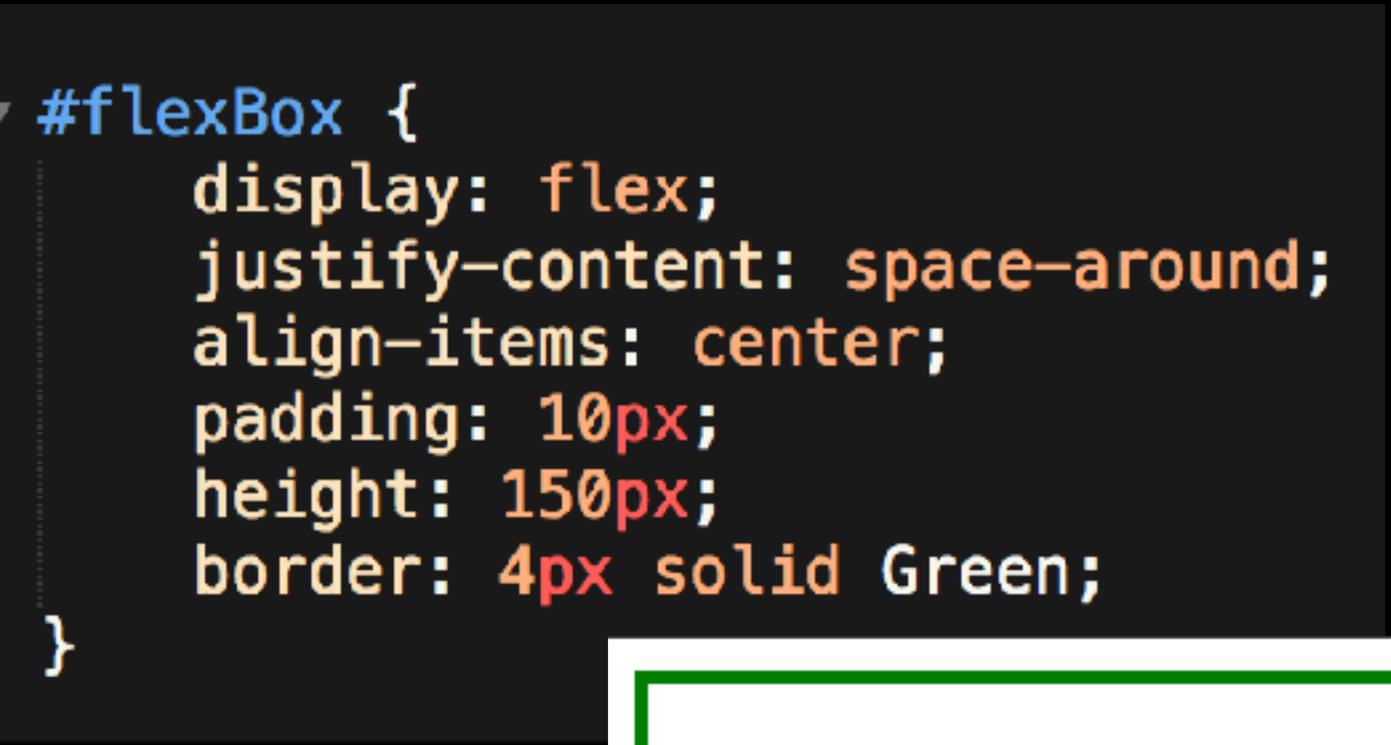


```
#flexBox {
    display: flex;
    justify-content: space-between;
    align-items: center;
    padding: 10px;
    height: 150px;
    border: 4px solid Green;
}
```

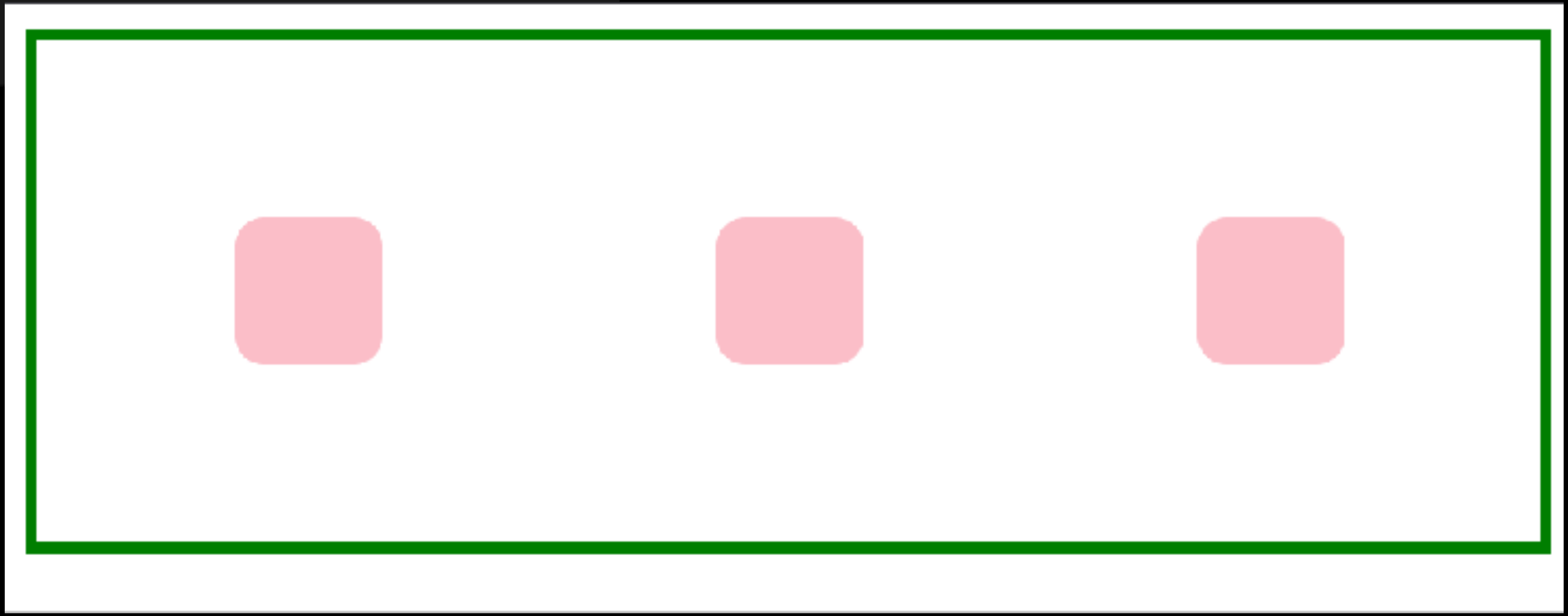


Flex Basics:

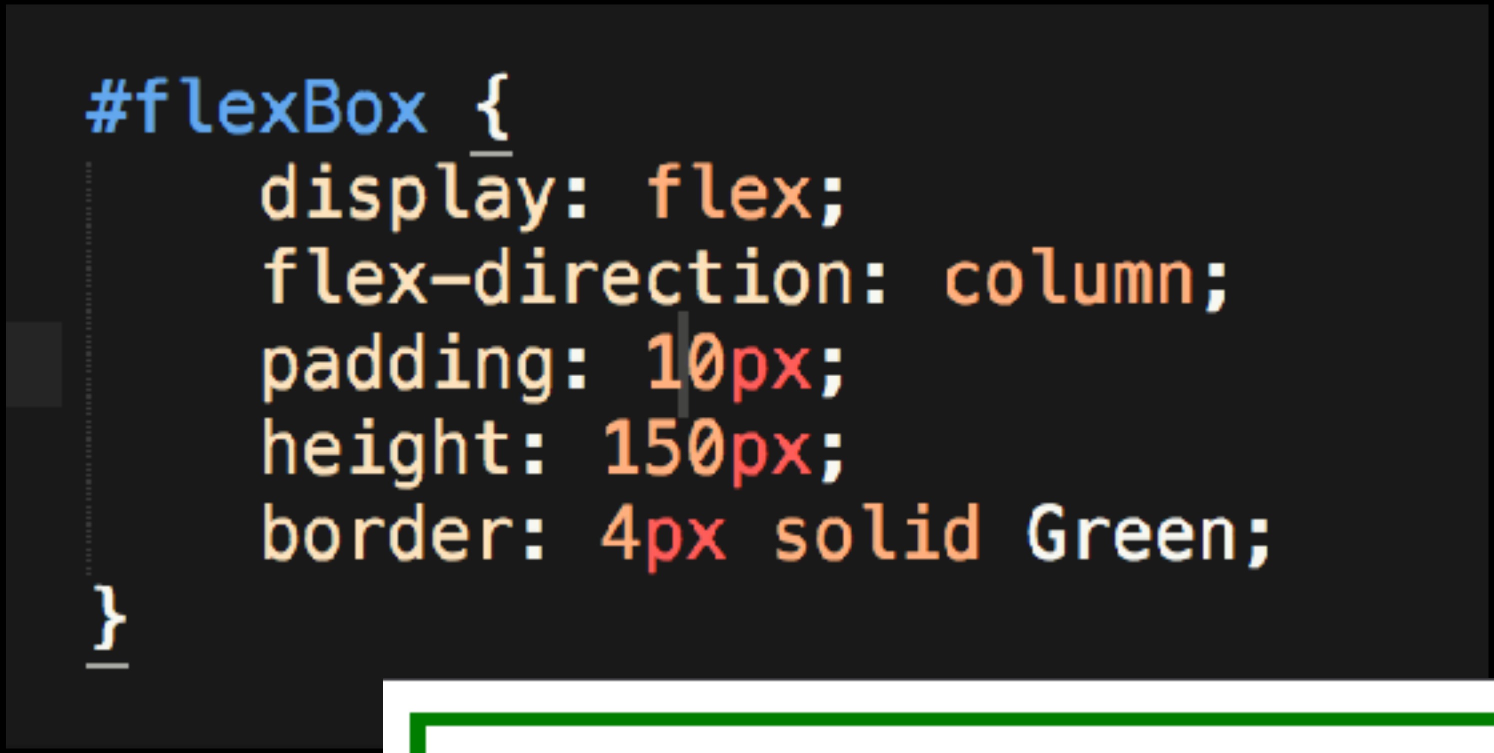
space-between + space-around



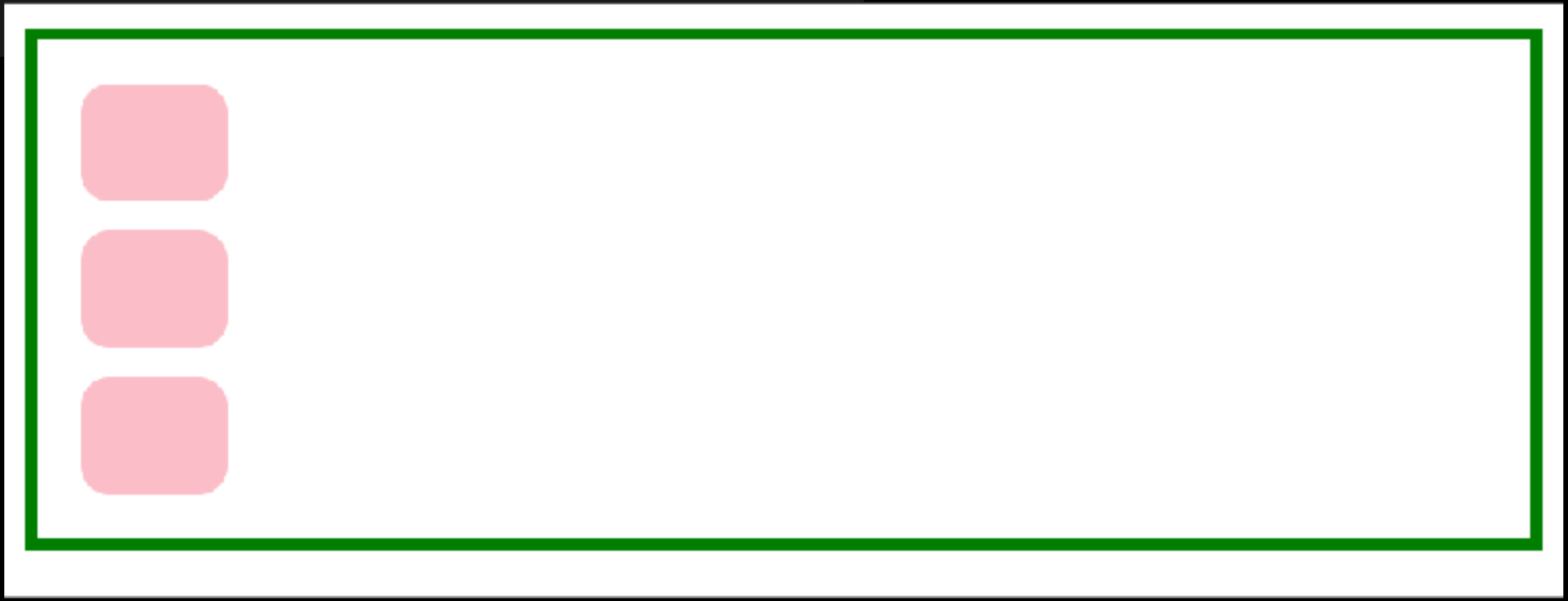
```
#flexBox {
    display: flex;
    justify-content: space-around;
    align-items: center;
    padding: 10px;
    height: 150px;
    border: 4px solid Green;
}
```



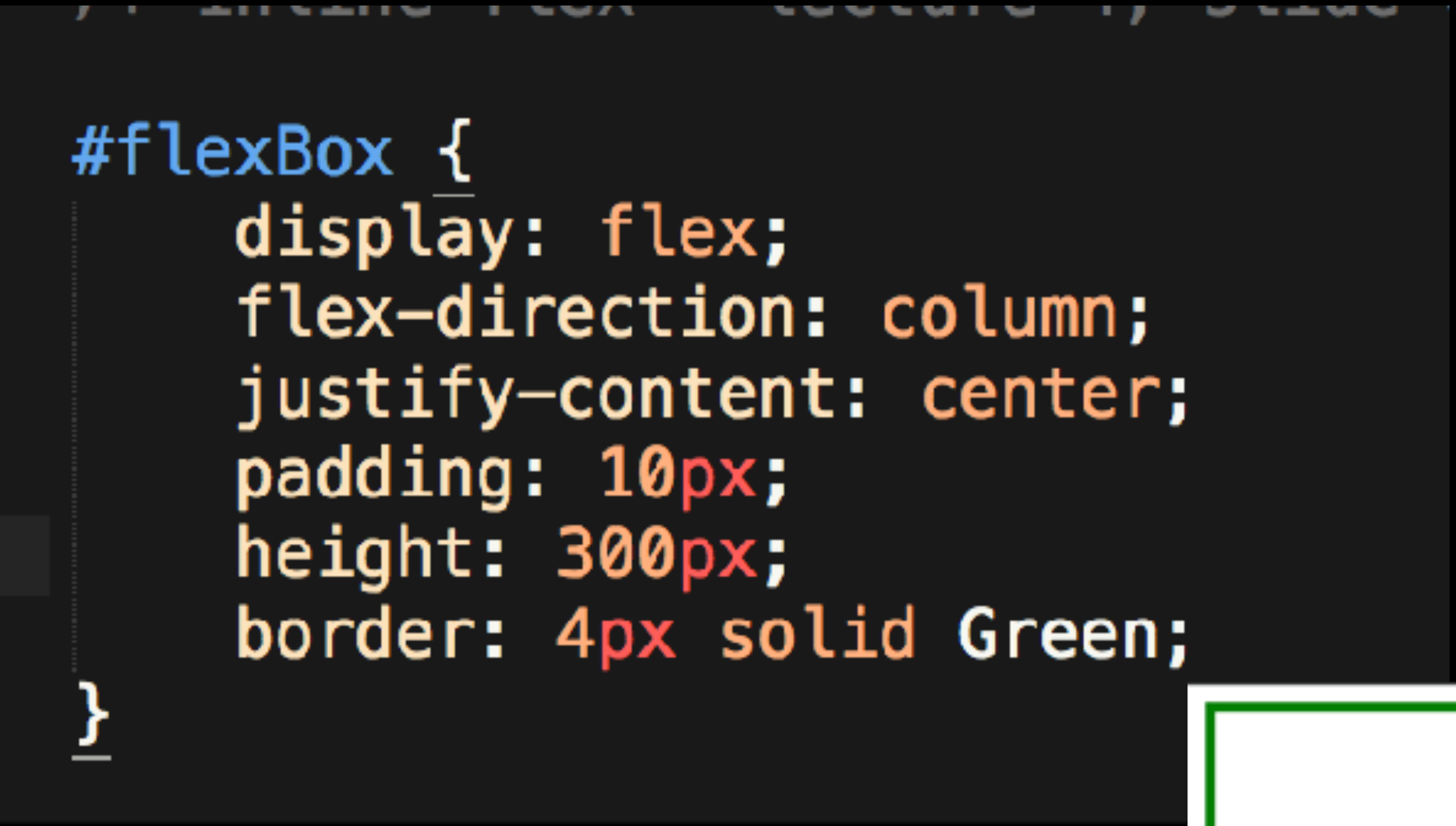
Flex Basics: flex-direction



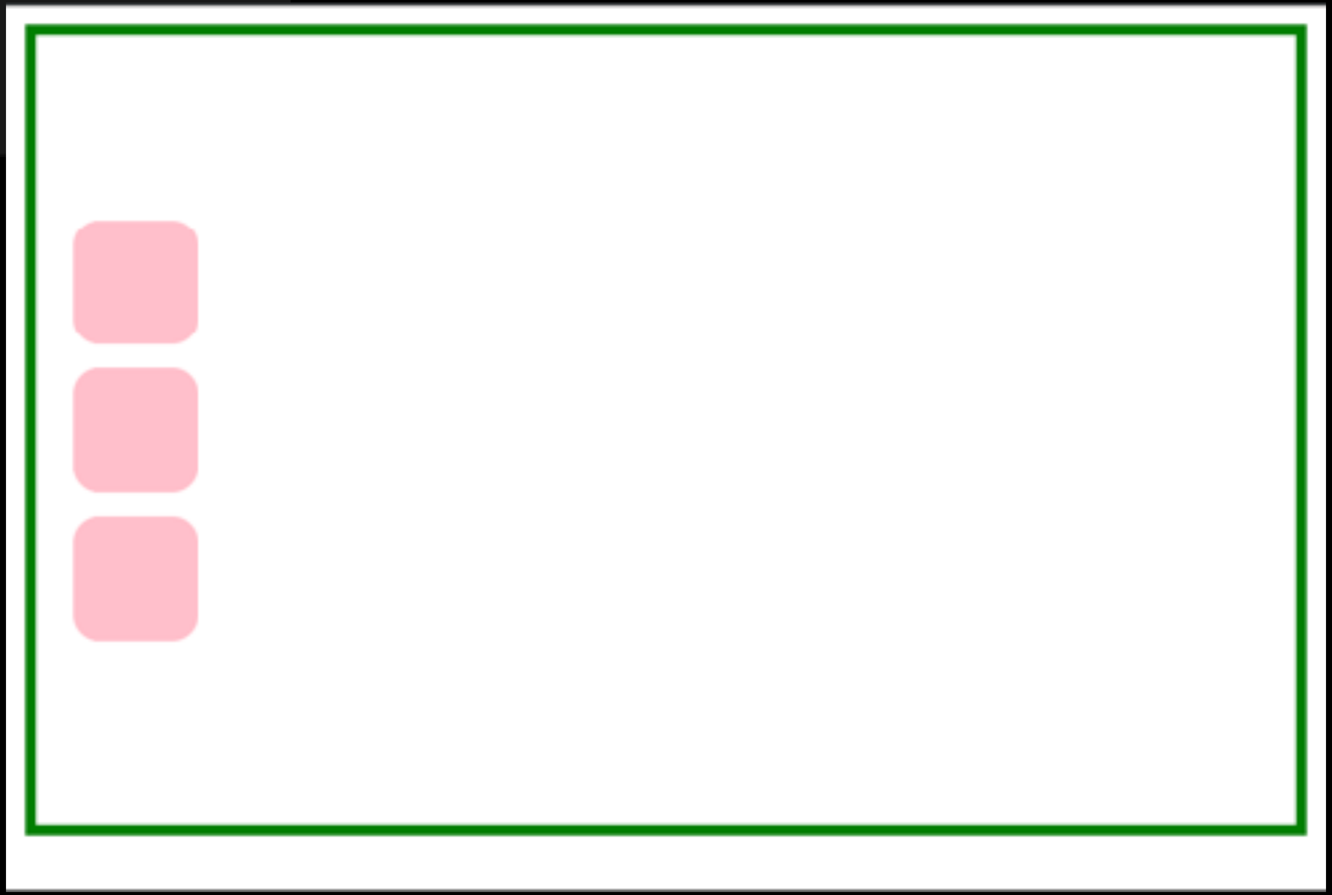
```css
#flexBox {
    display: flex;
    flex-direction: column;
    padding: 10px;
    height: 150px;
    border: 4px solid Green;
}
```



Flex Basics: flex-direction

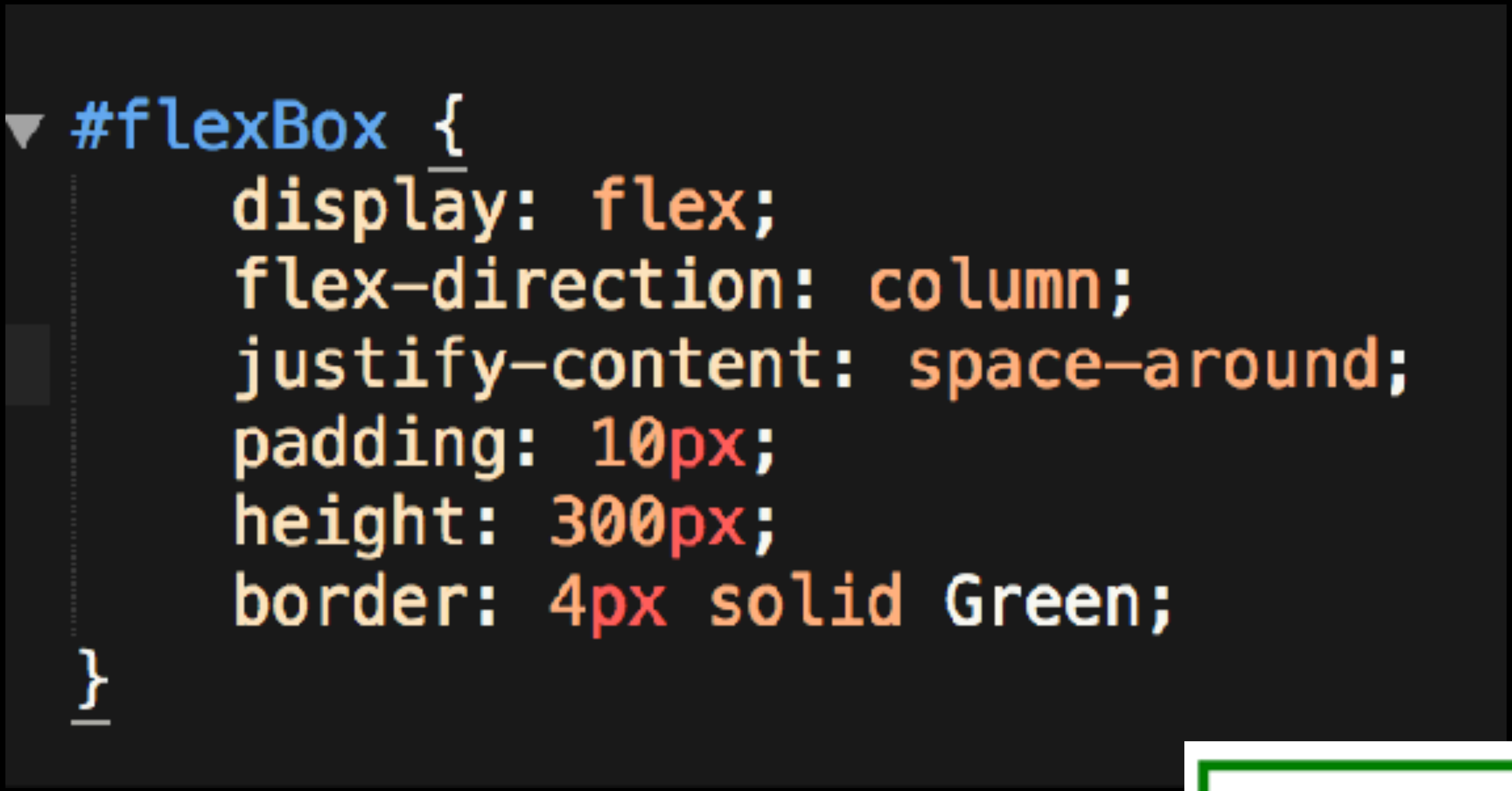


```
#flexBox {
    display: flex;
    flex-direction: column;
    justify-content: center;
    padding: 10px;
    height: 300px;
    border: 4px solid Green;
}
```



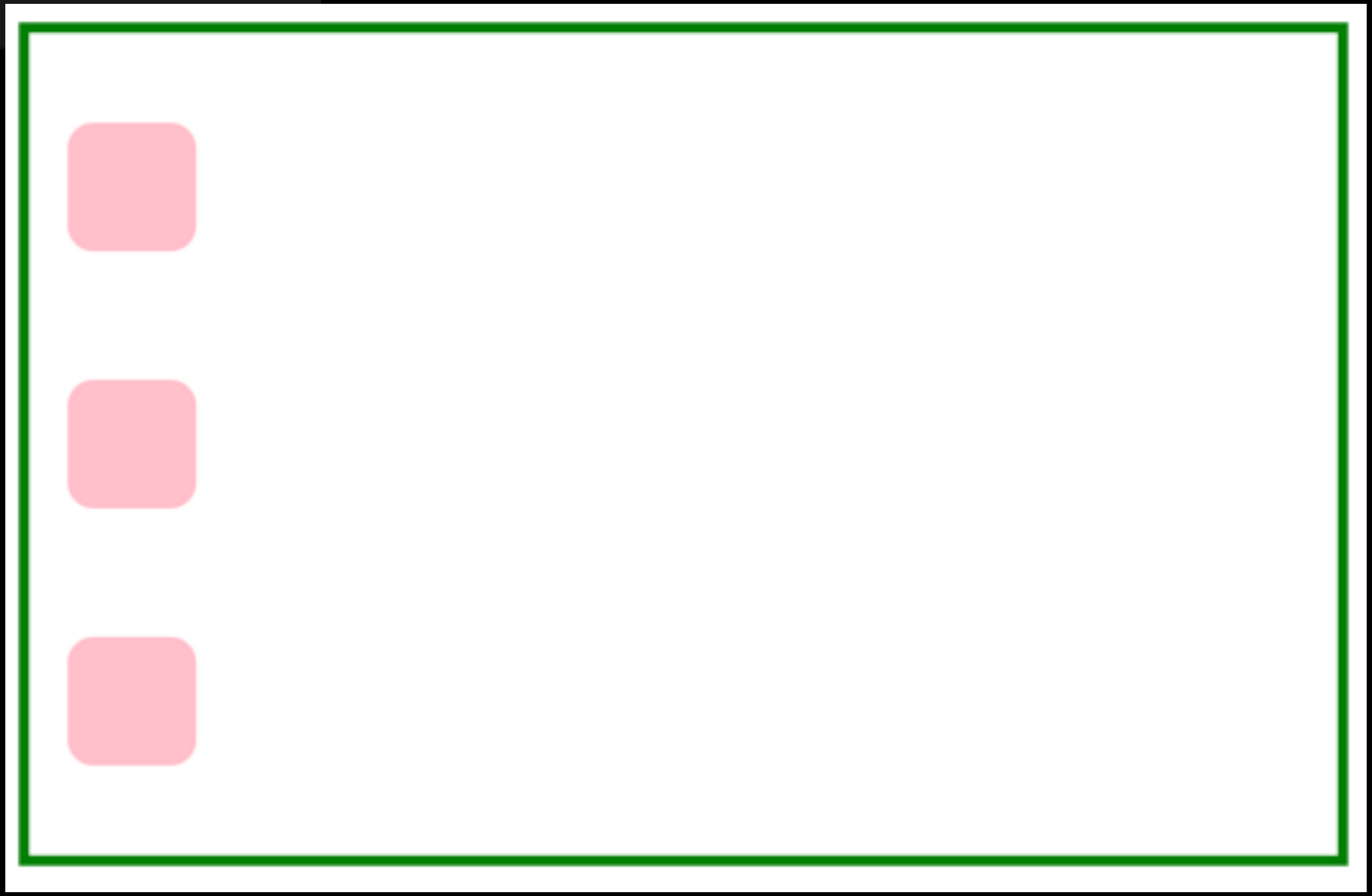
Now **justify-content** controls where the column is vertically in the box.

Flex Basics: flex-direction



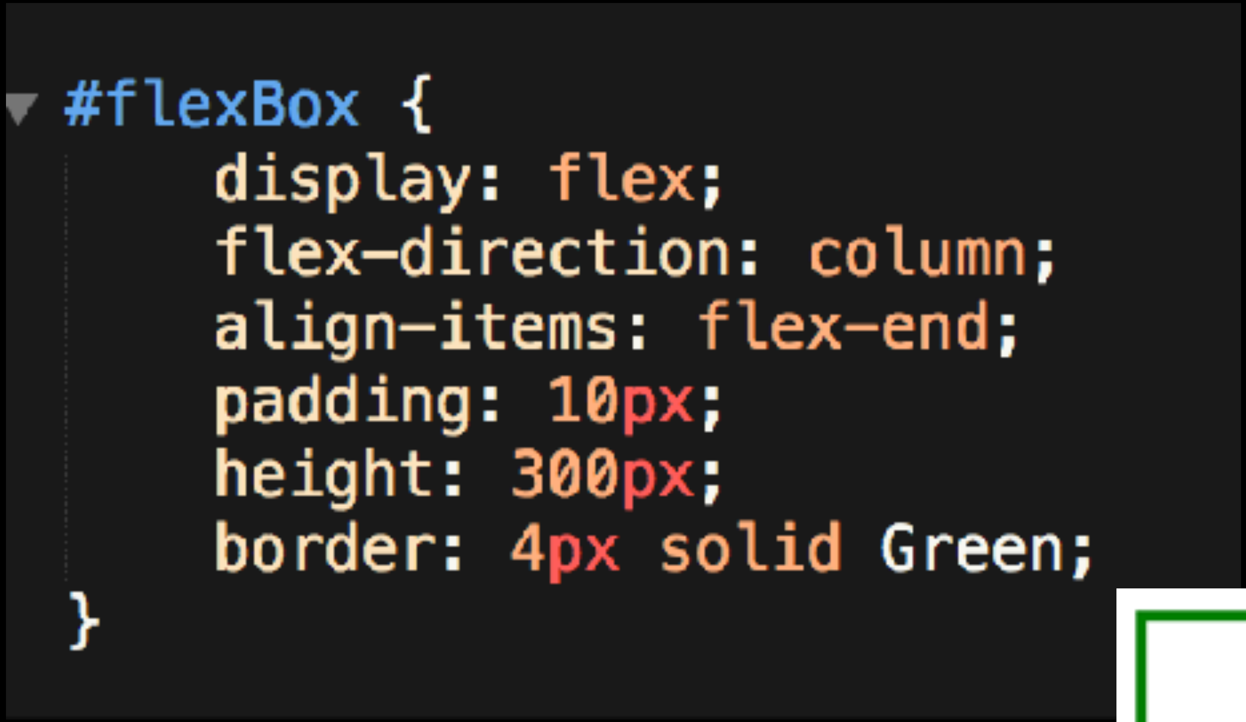
```
#flexBox {
    display: flex;
    flex-direction: column;
    justify-content: space-around;
    padding: 10px;
    height: 300px;
    border: 4px solid Green;
}
```

And you can also lay out columns instead of rows.



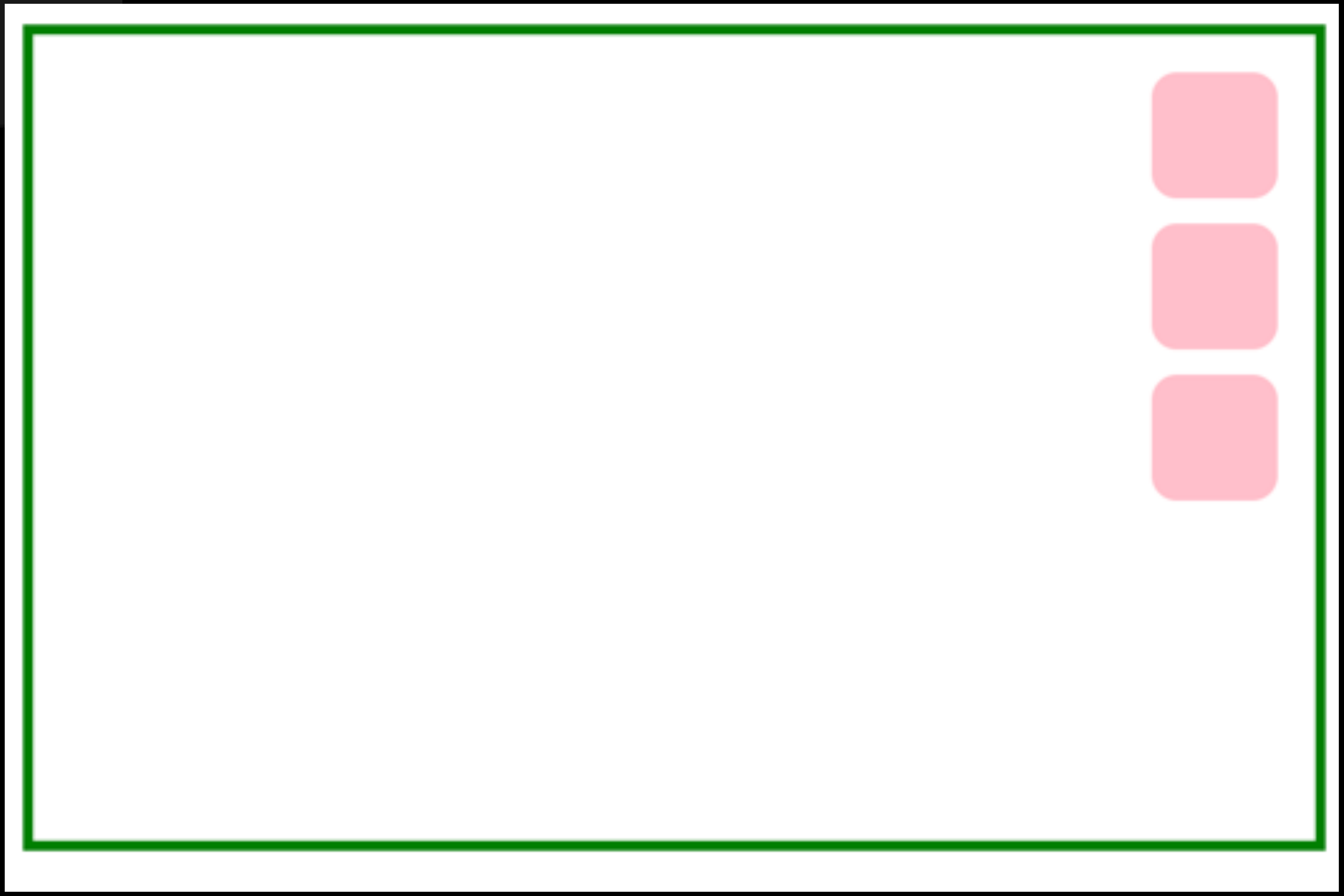
Now **justify-content** controls where the column is vertically in the box.

Flex Basics: flex-direction



```
#flexBox {
        display: flex;
        flex-direction: column;
        align-items: flex-end;
        padding: 10px;
        height: 300px;
        border: 4px solid Green;
}
```

And you can also lay out columns instead of rows.



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

Flex - different rendering model

When you set a container to **display: flex**, the direct children in that container are **flex items** + follow a new set of rules.

Flex items are not block or inline; they have different rules for their height, width + layout.

- The **contents** of a flex item follow the usual block/inline rules, relative to the flex item's boundary.

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

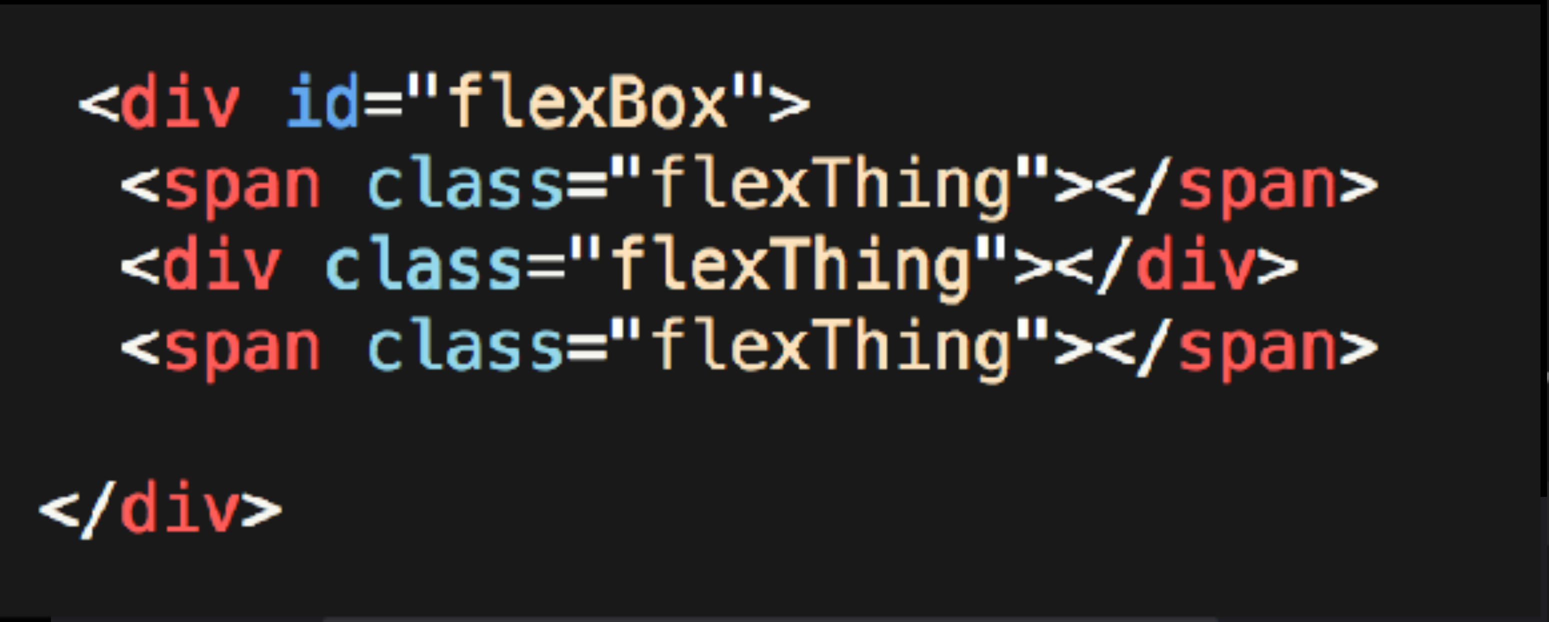
This initial width* of the flex item is called the **flex basis**.

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.

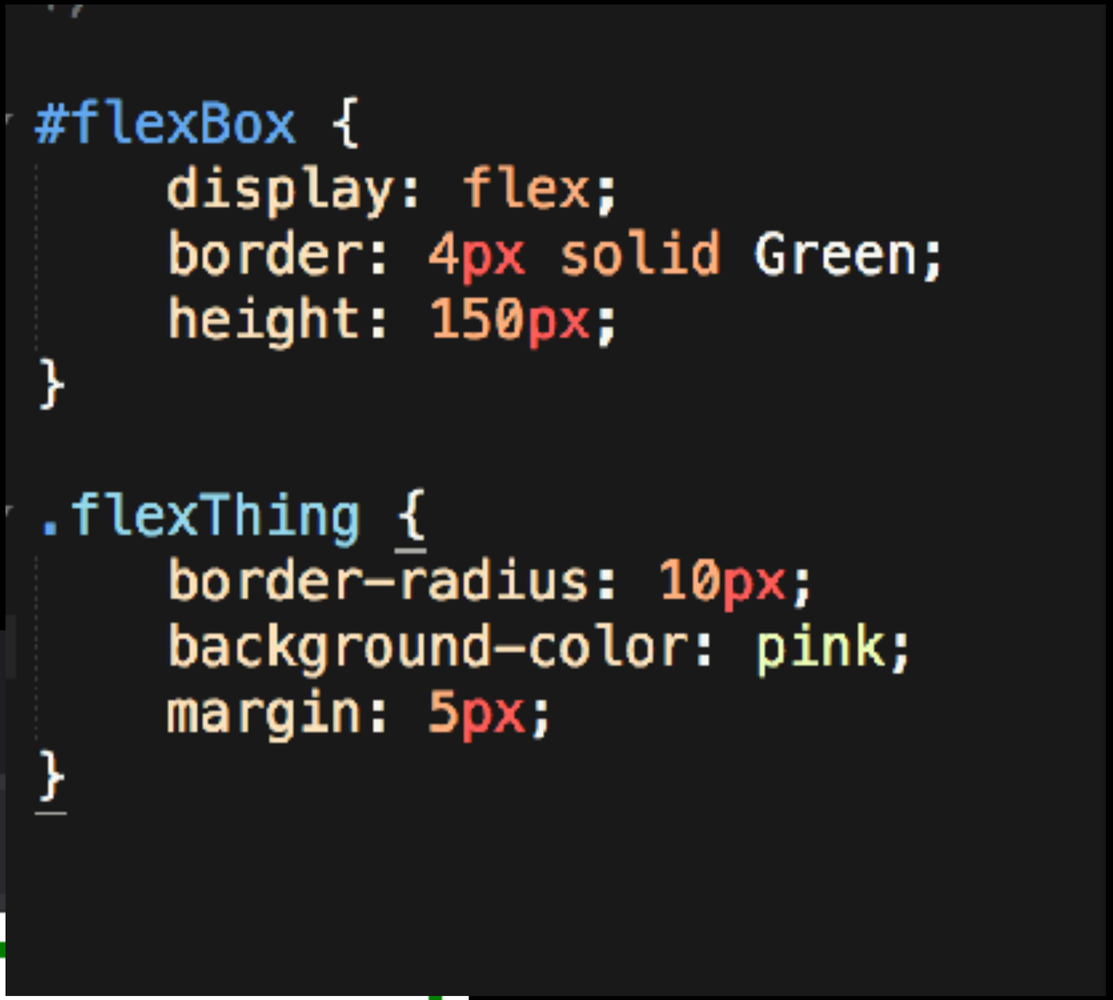
*width in the case of rows; height in the case of columns

Flex Basis

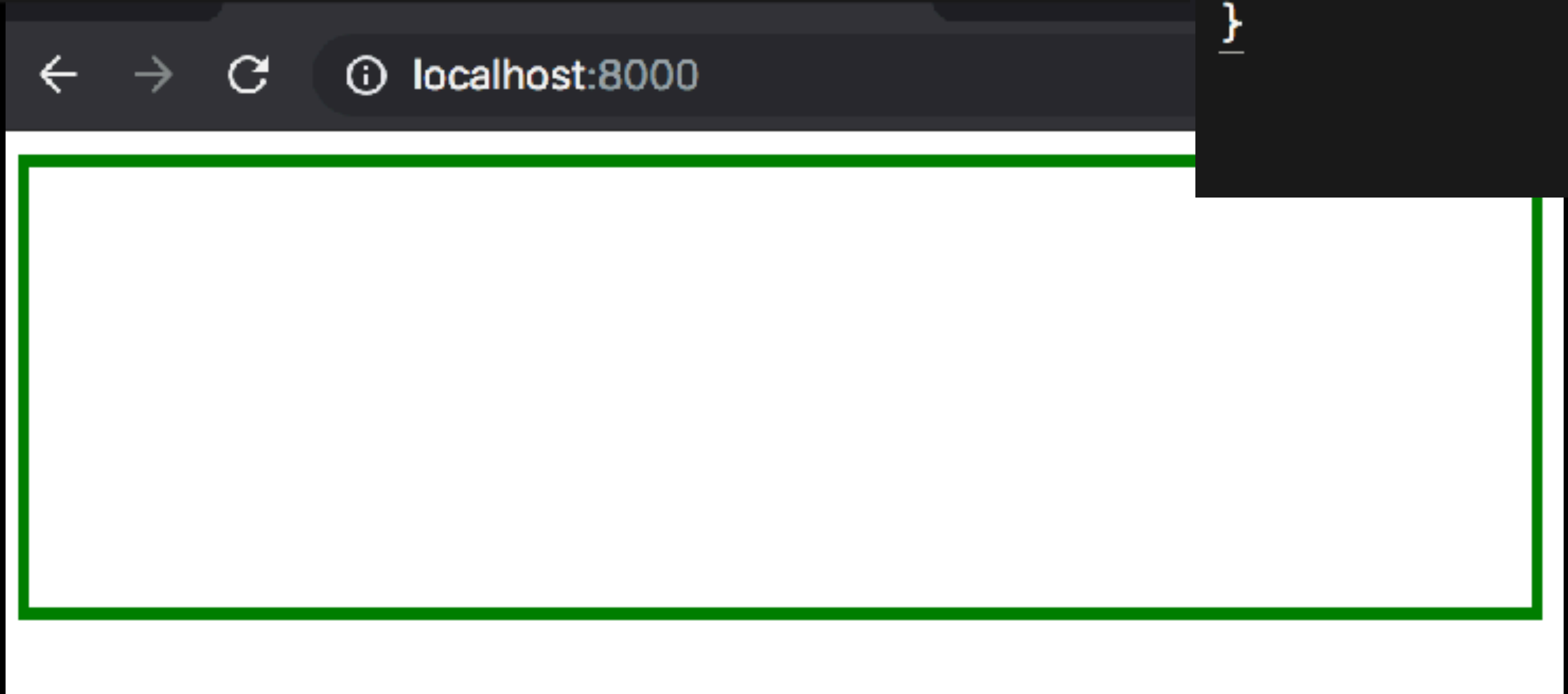
If we unset the height and width, our flex items disappears, because the **flex basis** is now the content size, which is empty:



```
<div id="flexBox">
 <span class="flexThing"></span>
 <div class="flexThing"></div>
 <span class="flexThing"></span>

</div>
```



```
#flexBox {
    display: flex;
    border: 4px solid Green;
    height: 150px;
}

.flexThing {
    border-radius: 10px;
    background-color: pink;
    margin: 5px;
}
```



flex-shrink

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

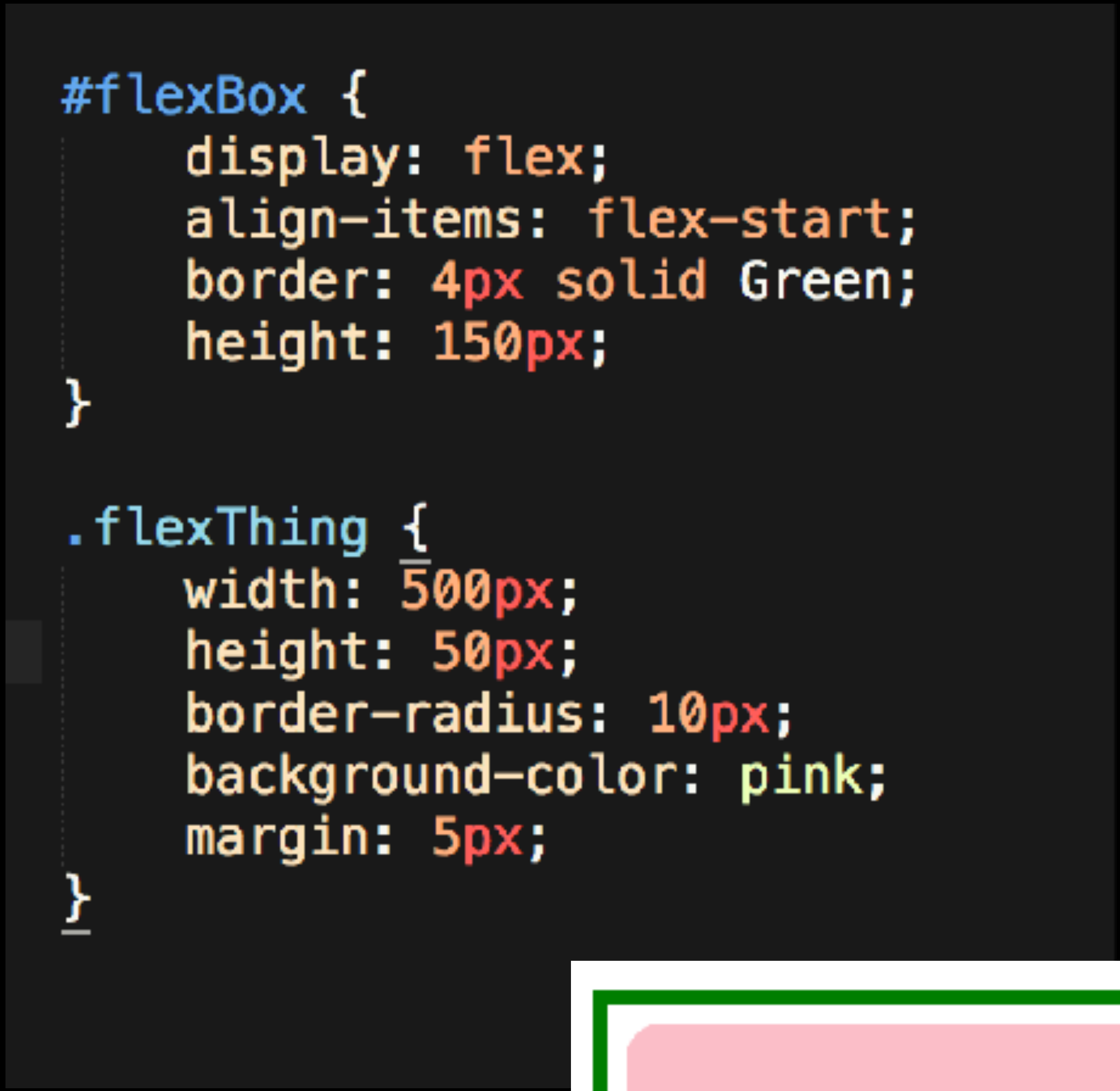
flex-shrink:

- 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**.

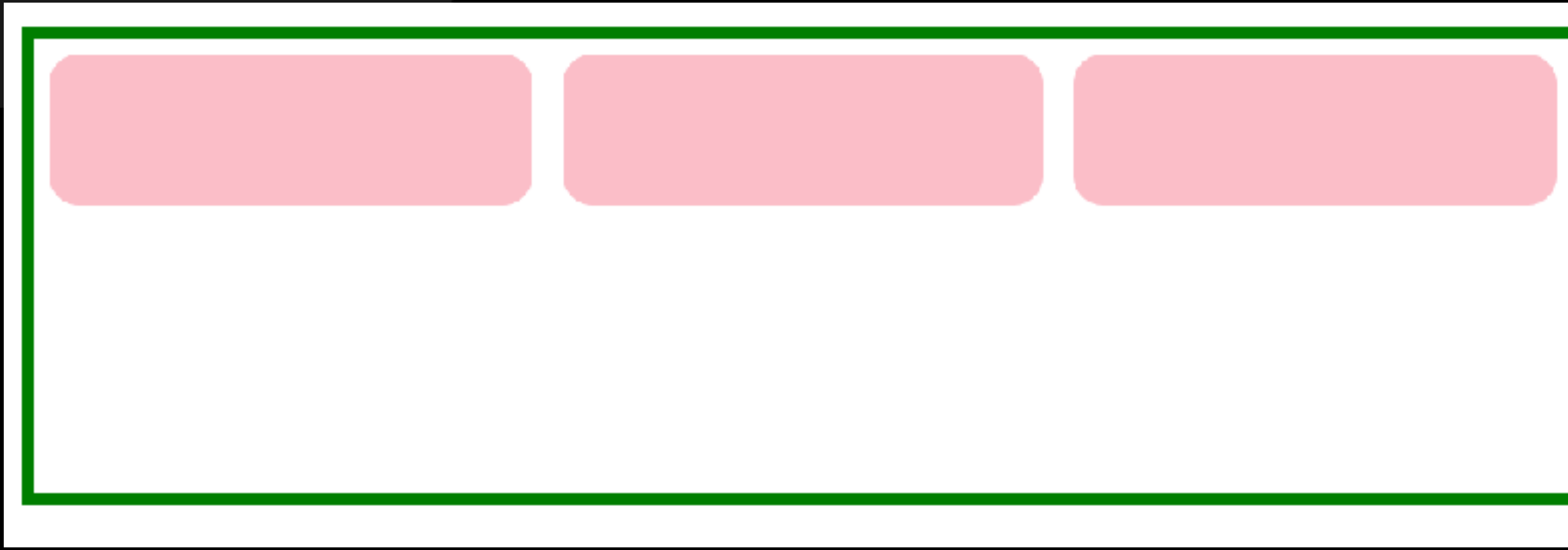
*width in the case of rows;
height in the case of columns

flex-shrink

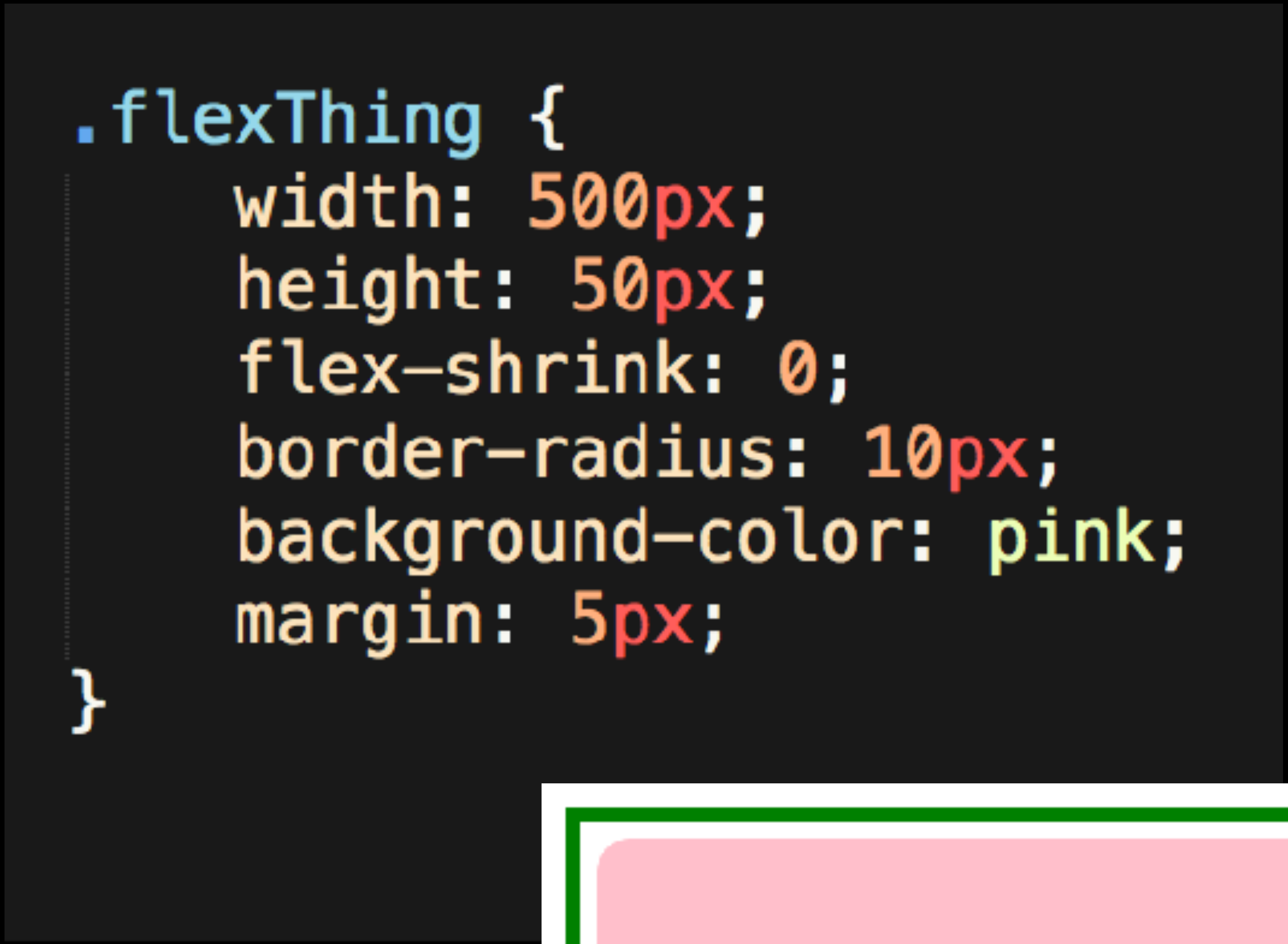


```
#flexBox {
    display: flex;
    align-items: flex-start;
    border: 4px solid Green;
    height: 150px;
}

.flexThing {
    width: 500px;
    height: 50px;
    border-radius: 10px;
    background-color: pink;
    margin: 5px;
}
```

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

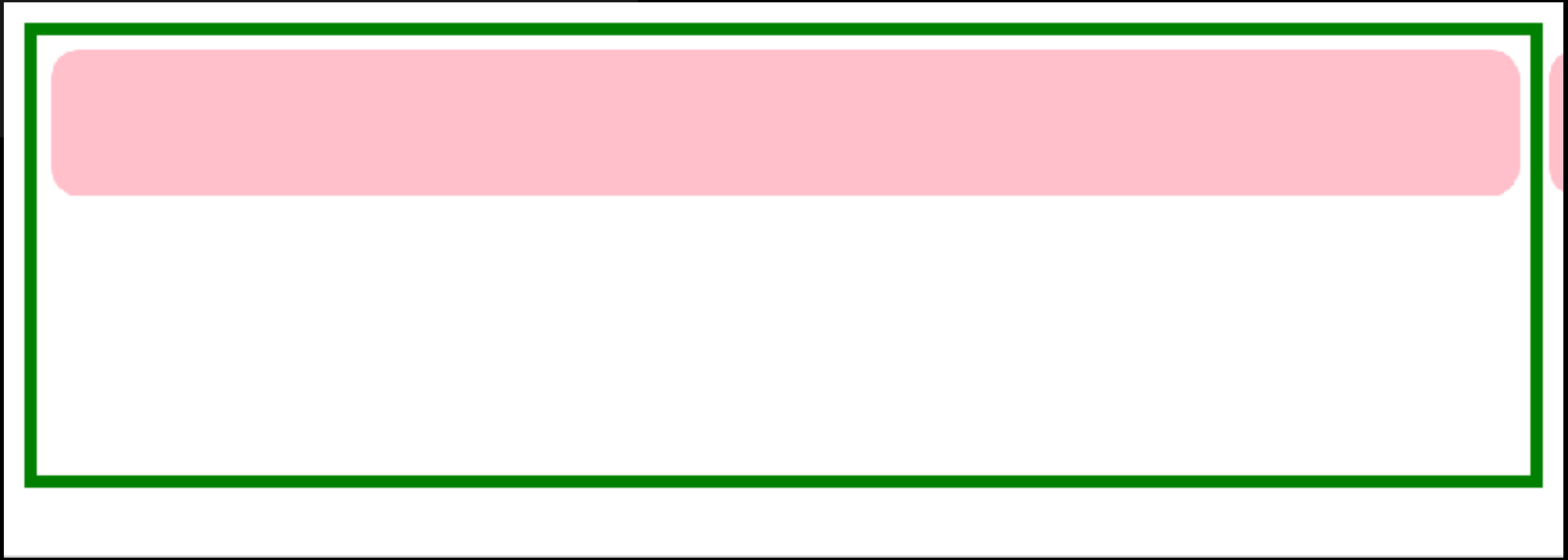


flex-shrink



```
.flexThing {
    width: 500px;
    height: 50px;
    flex-shrink: 0;
    border-radius: 10px;
    background-color: pink;
    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:

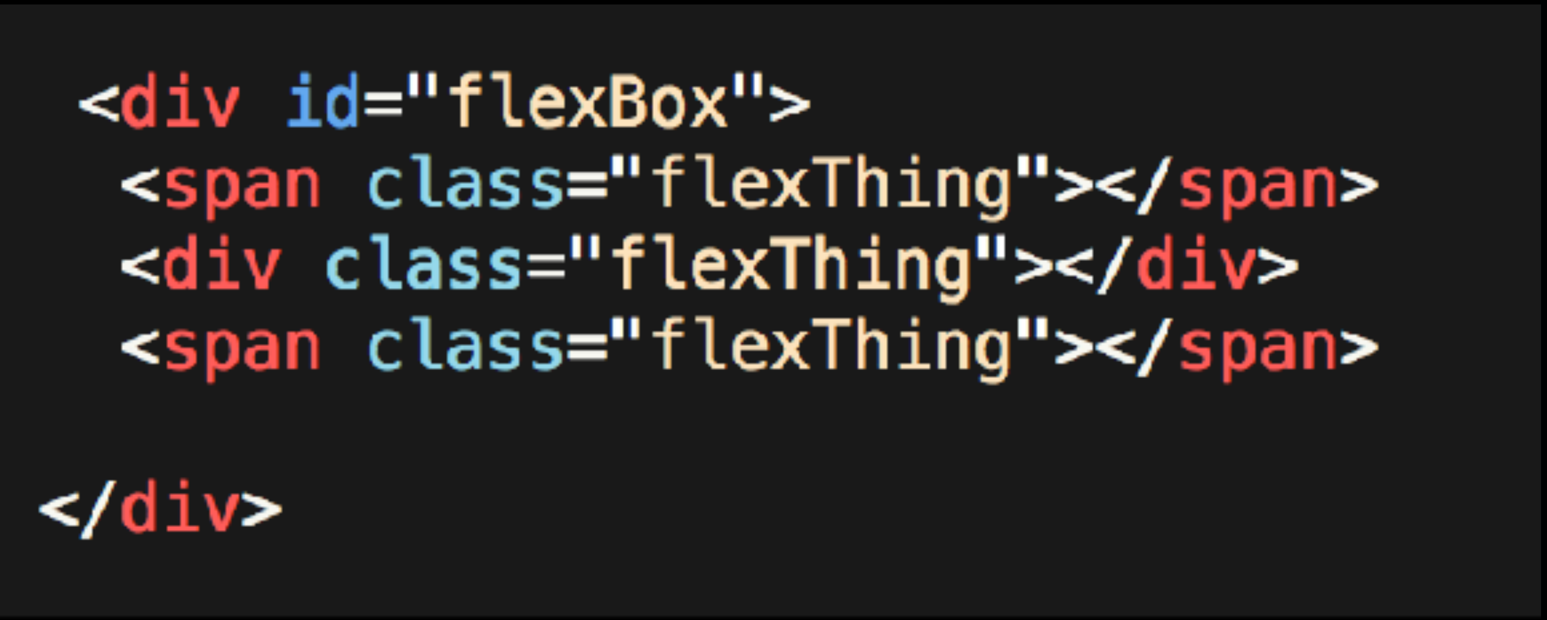
- If set to 1, the flex item grows itself as large as it can in the space remaining
- If set to 0, the flex item does not grow

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

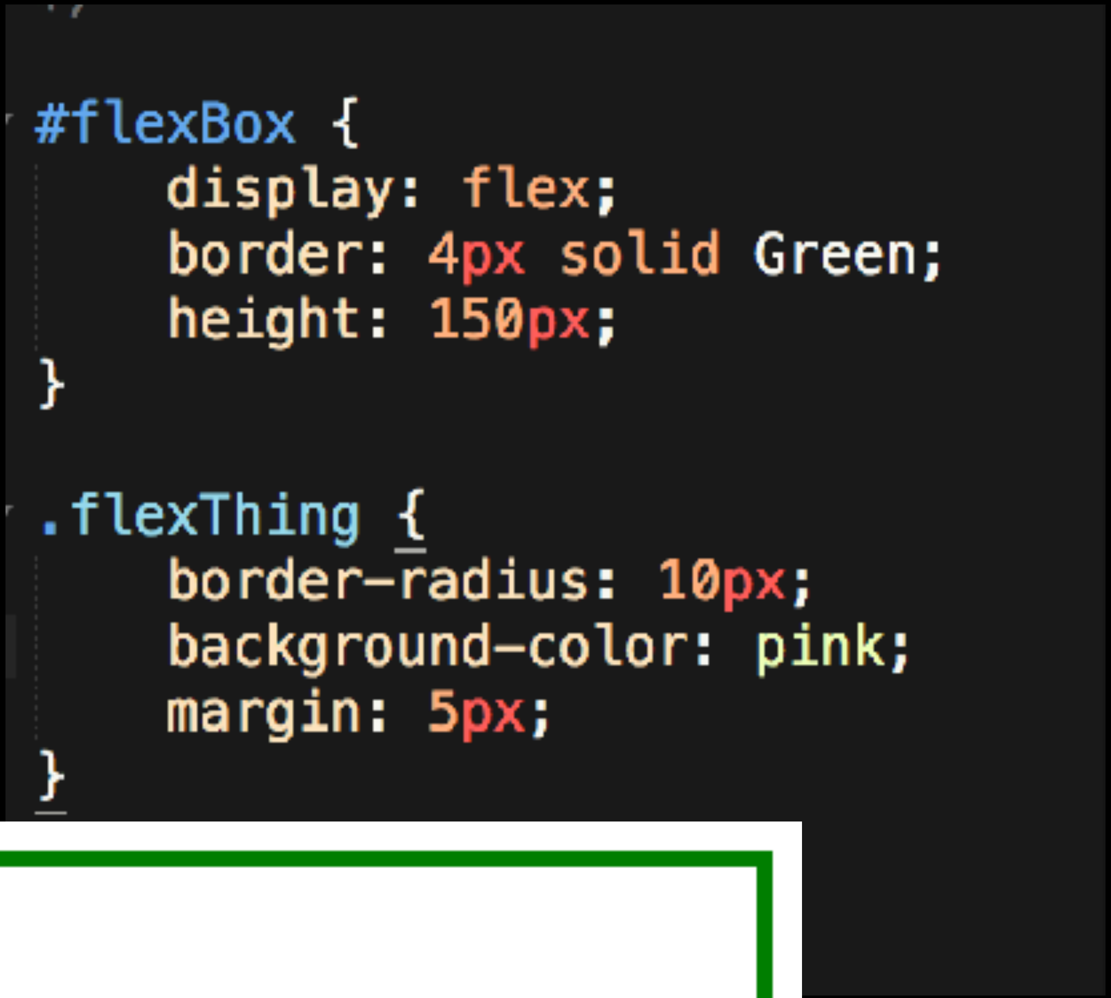
*width in the case of rows;
height in the case of columns

flex-grow

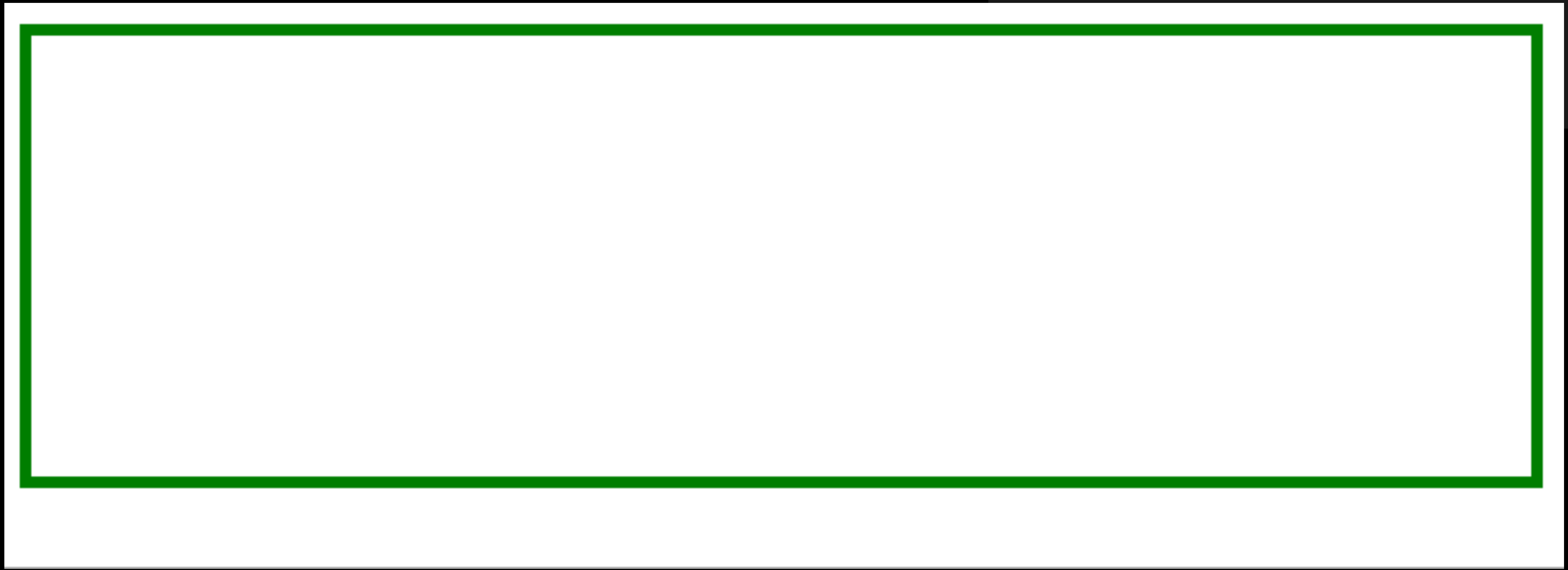
Let's unset the height + width of our flex items again.



```html
<div id="flexBox">
  <span class="flexThing"></span>
  <div class="flexThing"></div>
  <span class="flexThing"></span>

</div>
```

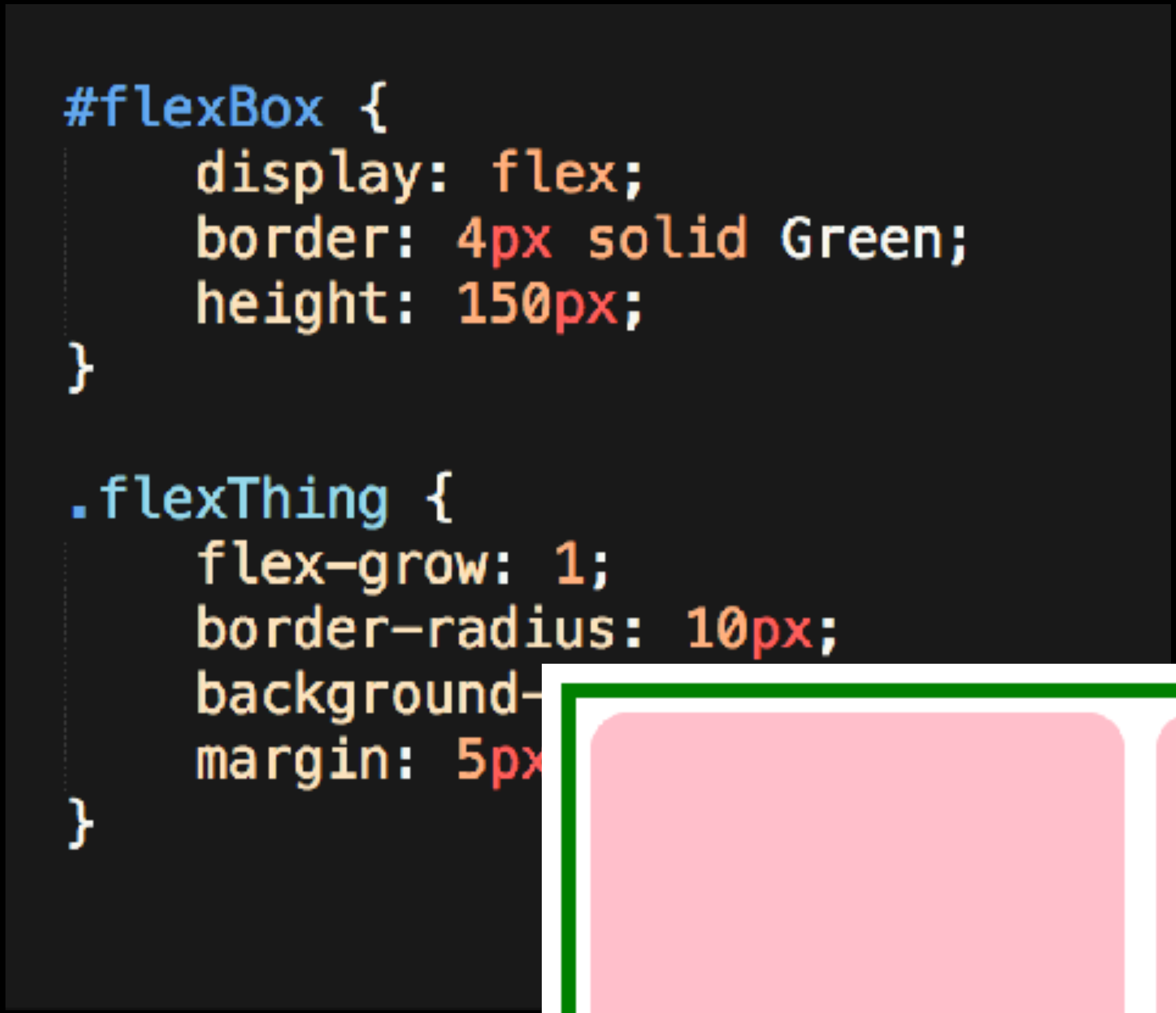


```css
#flexBox {
    display: flex;
    border: 4px solid Green;
    height: 150px;
}

.flexThing {
    border-radius: 10px;
    background-color: pink;
    margin: 5px;
}
```

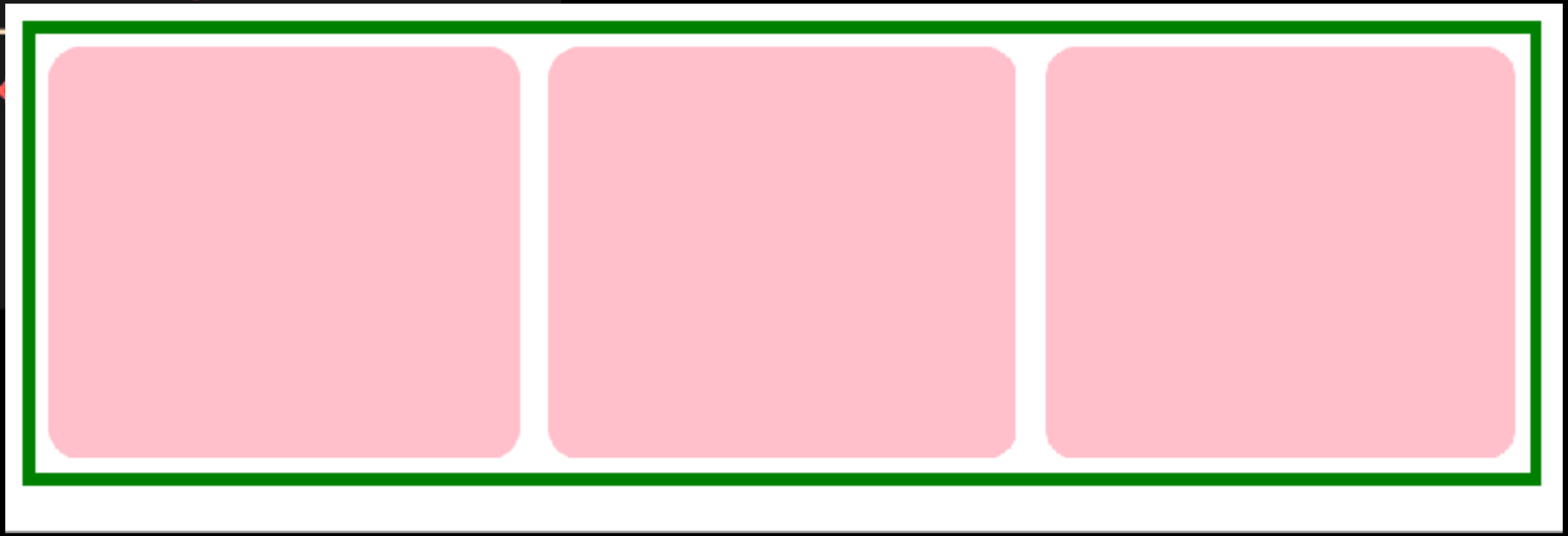


flex-grow

if we set **flex-grow**: 1;
the flex items fill the empty space.



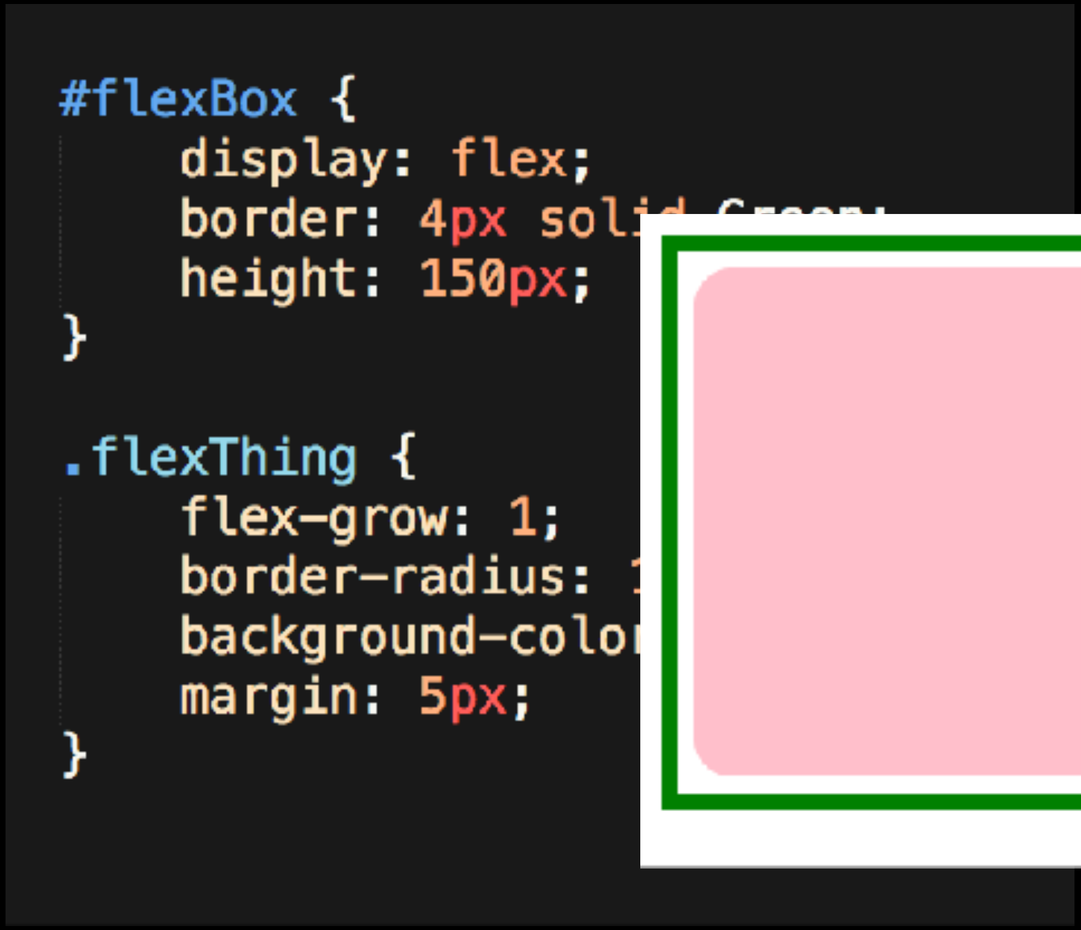
```
#flexBox {
    display: flex;
    border: 4px solid Green;
    height: 150px;
}

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



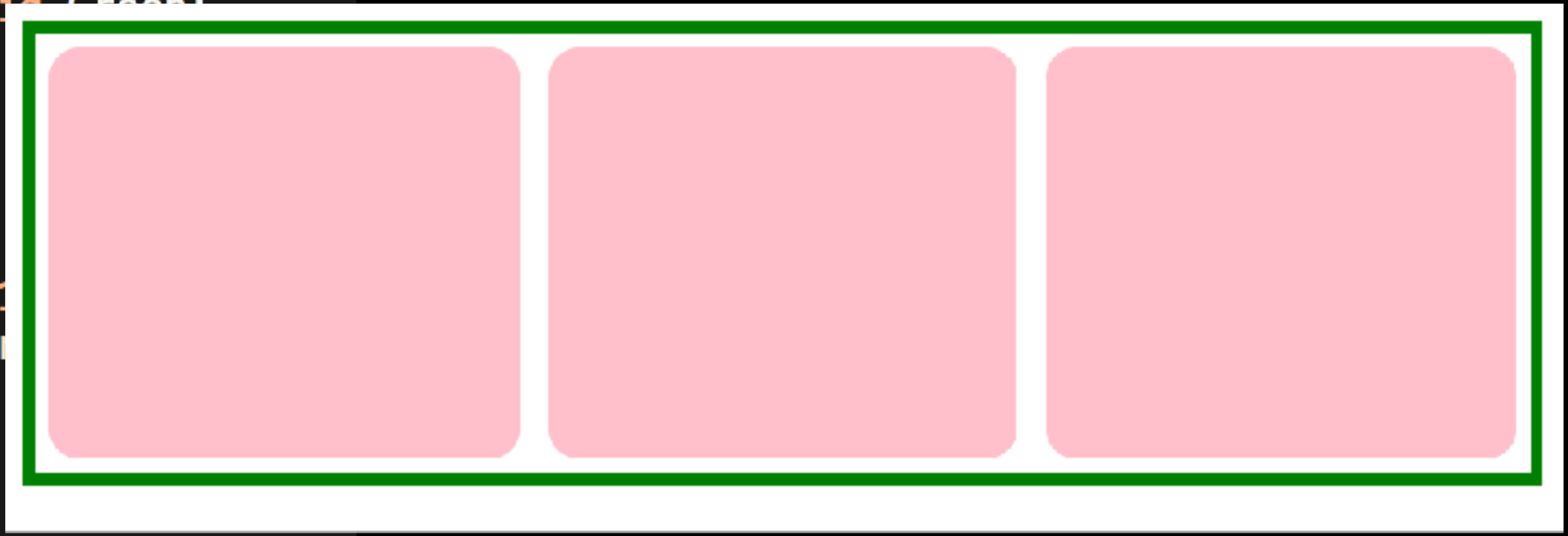
flex item height**?

note that **flex-grow** only controls width*

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



```
#flexBox {
    display: flex;
    border: 4px soli
    height: 150px;
}

.flexThing {
    flex-grow: 1;
    border-radius: 1
    background-colo
    margin: 5px;
}
```

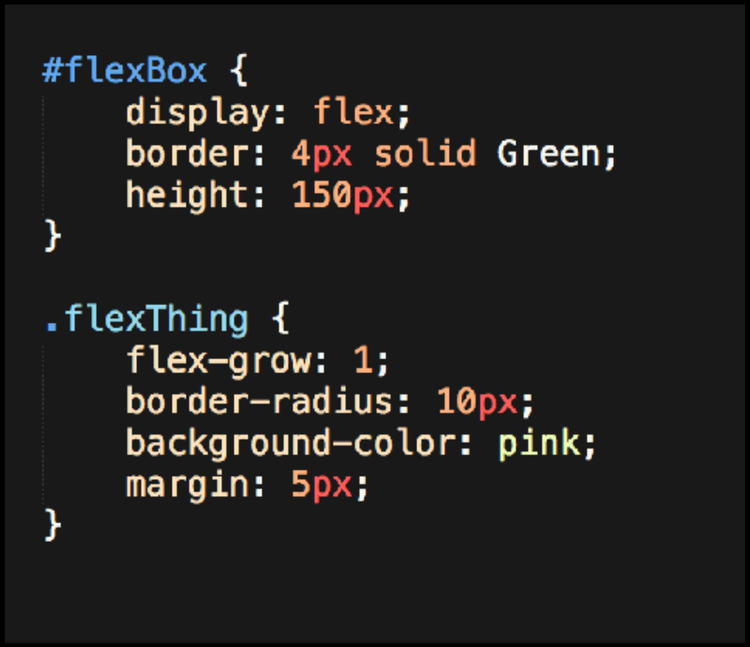


*width in the case of rows; height in the case of columns
**height in the case of rows; width in the case of columns

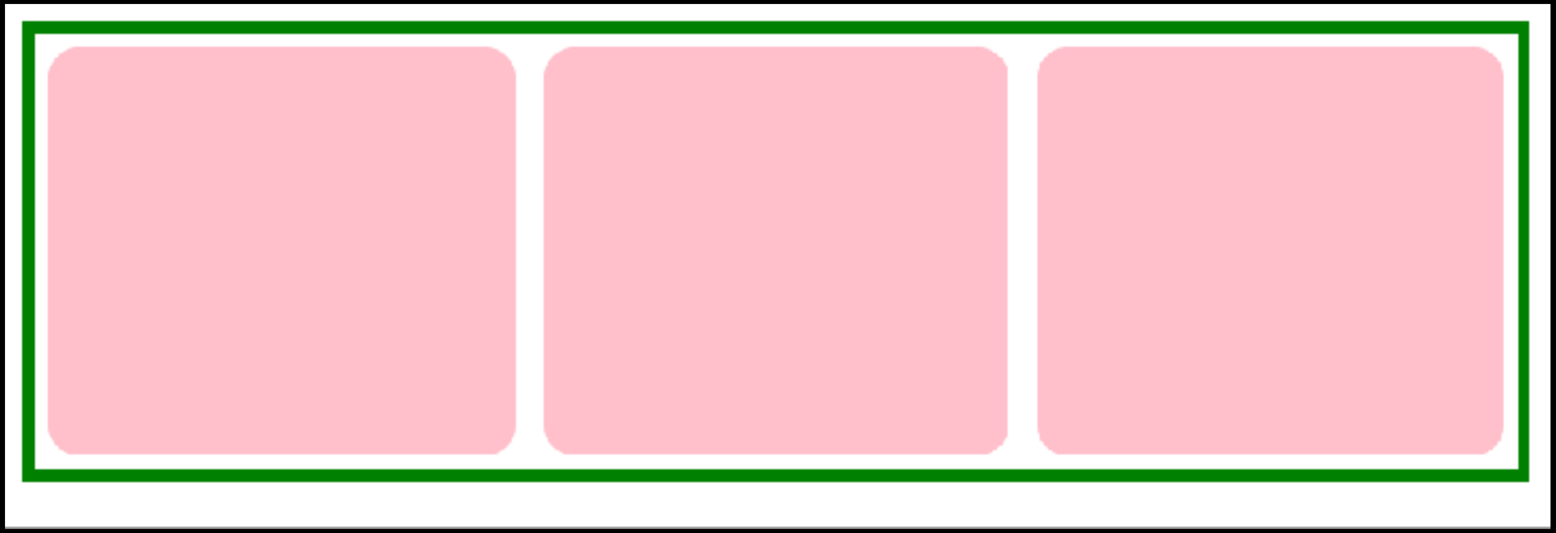
align-items: stretch;

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)



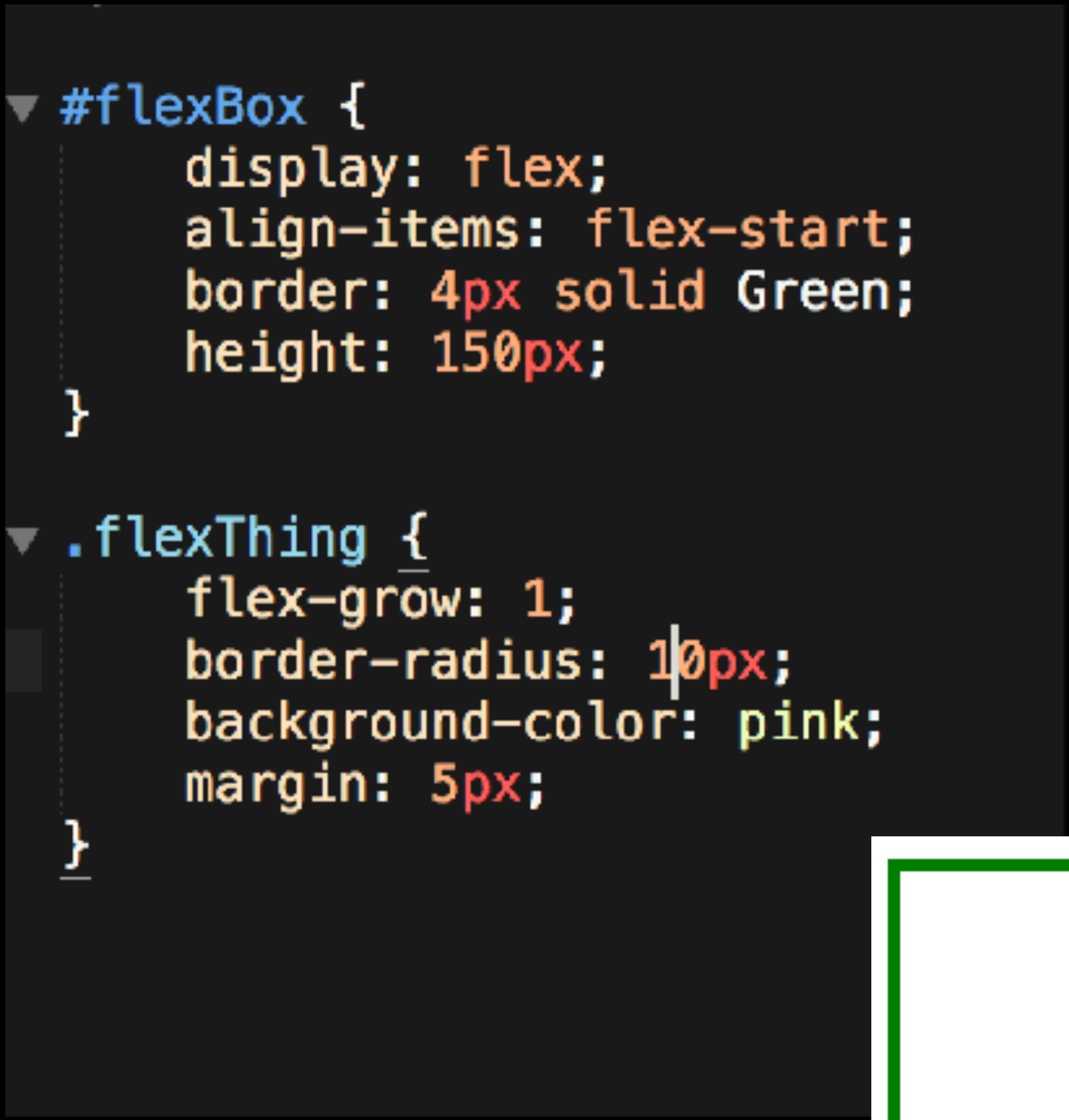
```
#flexBox {
    display: flex;
    border: 4px solid Green;
    height: 150px;
}

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



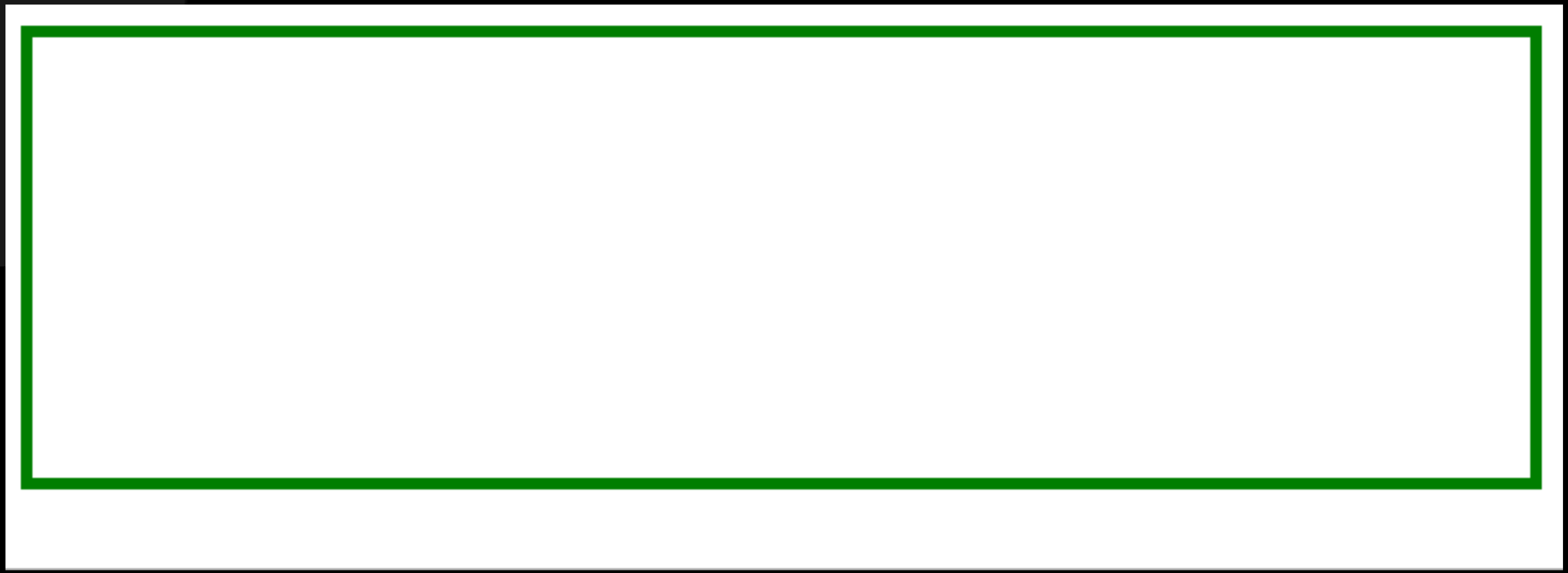
*veritcally in the case of rows; horizontally in the case of columns

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:



```
#flexBox {
    display: flex;
    align-items: flex-start;
    border: 4px solid Green;
    height: 150px;
}

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



CSS TRANSITIONS

CSS Transitions smooth out otherwise abrupt changes to property values between two states over time by filling in the frames in between. Animators call that tweening. When used with reserve, CSS Transitions can add sophistication and polish to your interfaces and even improve usability.

When applying a transition, you have a few decisions to make, each of which is set with a CSS property:

- Which CSS property to change (**transition-property**) (Required)
- How long it should take (**transition-duration**) (Required)
- The manner in which the transition accelerates (**transition-timing- function**)
- Whether there should be a pause before it starts (**transition-delay**)

Transitions require a **beginning state** and an **end state**. The element as it appears when it first loads is the beginning state. The end state needs to be triggered by a state change such as :hover, :focus, or :active…

CSS Animation Selectors

: hover

: focus

: active

transition-property

identifies the CSS property that is changing and that you want to transition smoothly. In our example, it’s the background-color. You can also change the foreground color, borders, dimensions, font- and text-related attributes, and many more. TABLE 18-1 lists the animatab CSS properties as of this writing. The general rule is that if its value is a color, length, or number, that property can be a transition property.

Backgrounds

background-color
background-position

Borders and outlines

border-bottom-color
border-bottom-width
border-left-color
border-left-width
border-right-color
border-right-width
border-top-color
border-top-width
border-spacing
outline-color
outline-width

Color and opacity

color
opacity
visibility

Font and text

font-size
font-weight
letter-spacing
line-height
text-indent
text-shadow
word-spacing
vertical-align

Position

top
right
bottom
left
z-index
clip-path

Transforms

transform
transform-origin

Element box measurements

height
width
max-height
max-width
min-height
min-width
margin-bottom
margin-left
margin-top
padding-bottom
padding-left
padding-right
padding-top

Animateable CSS Properties

Timing Functions

```
. thisAwesomeClass {
        transition-timing-function :

                ease

                linear

                ease-in

                ease-out

                ease-in-out
                step-start

                step-end

                steps

                cubic-bezier(#,#,#,#)

    }
```

the css property

possible values you can set

The **property** and the **duration** are required and form the foundation of a transition, but you can refine it further. There are a number of ways a **transition** can roll out over time.

For example, it could start out fast and then slow down, start out slow and speed up, or stay the same speed all the way through, just to name a few possibilities. I think of it as the transition "style," but in the spec, it is known as the timing function or easing function.

The timing function you choose can have a big impact on the feel and believability of the animation, so if you plan on using transitions and CSS animations, it is a good idea to get familiar with the options.

ease

Starts slowly, accelerates quickly, and then slows down at the end. This is the default value and works just fine for most short transitions.

linear

Stays consistent from the transition's beginning to end. Because it is so consistent, some say it has a mechanical feeling.

ease-in

Starts slowly, then speeds up.

ease-out

Starts out fast, then slows down.

ease-in-out

Starts slowly, speeds up, and then slows down again at the very end. It is similar to ease, but with less pronounced acceleration in the middle.

transition-delay

The transition-delay property, as you might guess, delays the start of the animation by a specified amount of time.

The Shorthand transition Property

The authors of the CSS3 spec had the good sense to give us the shorthand transition property to combine all of these properties into one declaration.You've seen this sort of thing with the shorthand border property. Here is the syntax:

```
transition: property duration timing-function delay;
```

```
.theClass {
    transition: background-color 0.3s ease-in-out 0.2s;
    }
```

The values for each of the **transition-*** properties are listed out, separated by character spaces. The order isn't important as long as the **duration** (which is required) appears before **delay** (which is optional). If you provide only one time value, it will be assumed to be the duration.

The sub-properties of the **animation** property are:

animation-delay	Configures the delay between the time the element is loaded and the beginning of the animation sequence.
animation-direction	Configures whether or not the animation should alternate direction on each run through the sequence or reset to the start point and repeat itself.
animation-duration	Configures the length of time that an animation should take to complete one cycle.
animation-iteration-count	Configures the number of times the animation should repeat; you can specify infinite to repeat the animation indefinitely.
animation-name	Specifies the name of the **@keyframes** at-rule describing the animation's keyframes.

animation-play-state	Lets you pause and resume the animation sequence.
animation-timing-function	Configures the timing of the animation; that is, how the animation transitions through keyframes, by establishing acceleration curves.
animation-fill-mode	Configures what values are applied by the animation before and after it is executing.

@keyframes + animation property

Programmed: Rules, Codes, and Choreographies in Art, 1965–2018

Rule, Instruction, Algorithm: Ideas as Form

Rule, Instruction, Algorithm: Generative Measures

Rule, Instruction, Algorithm: Collapsing Instruction + Form

Signal, Sequence, Resolution: Image Resequenced

Signal, Sequence, Resolution: Liberating the Signal

Signal, Sequence, Resolution: Realities Encoded