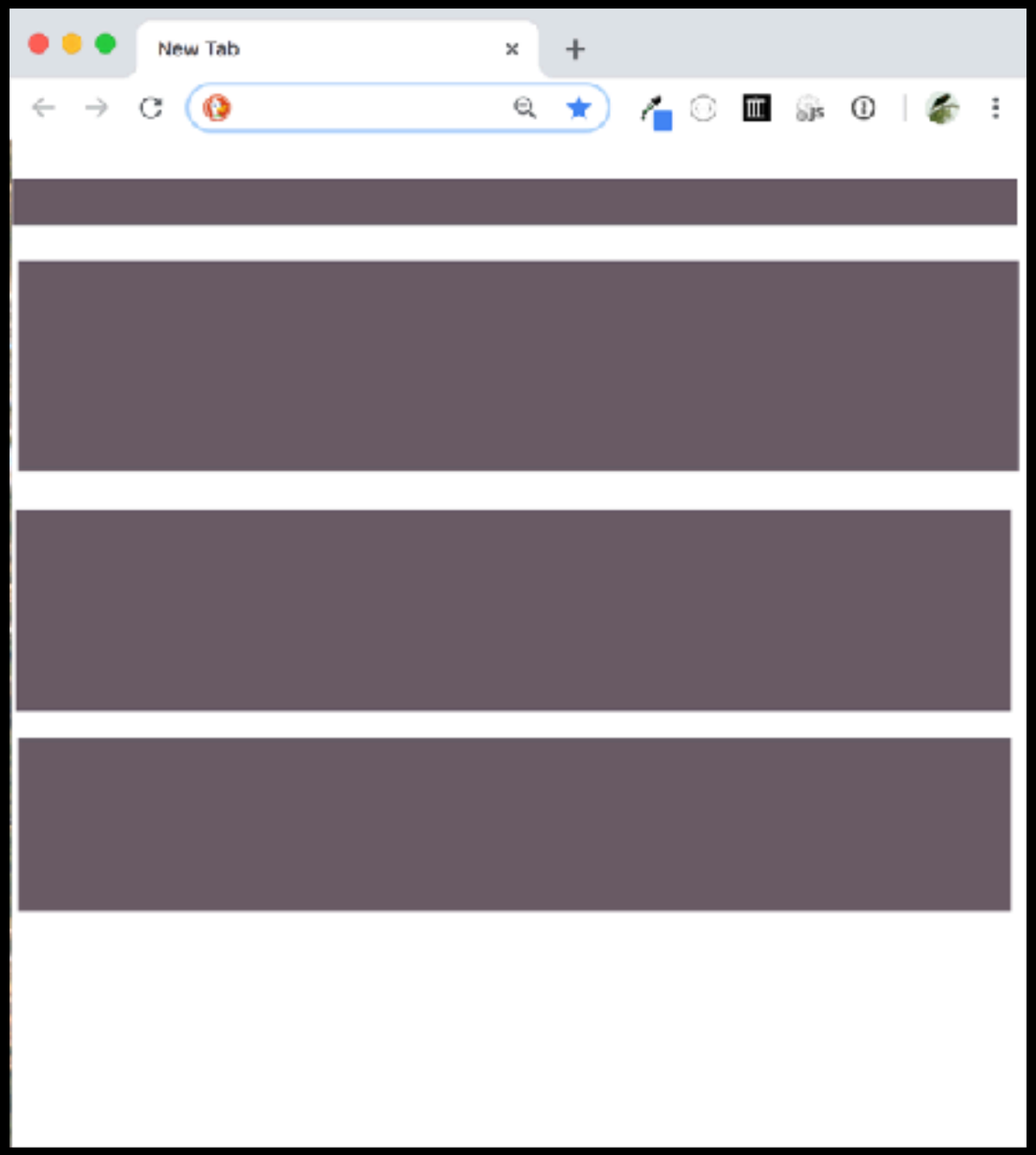
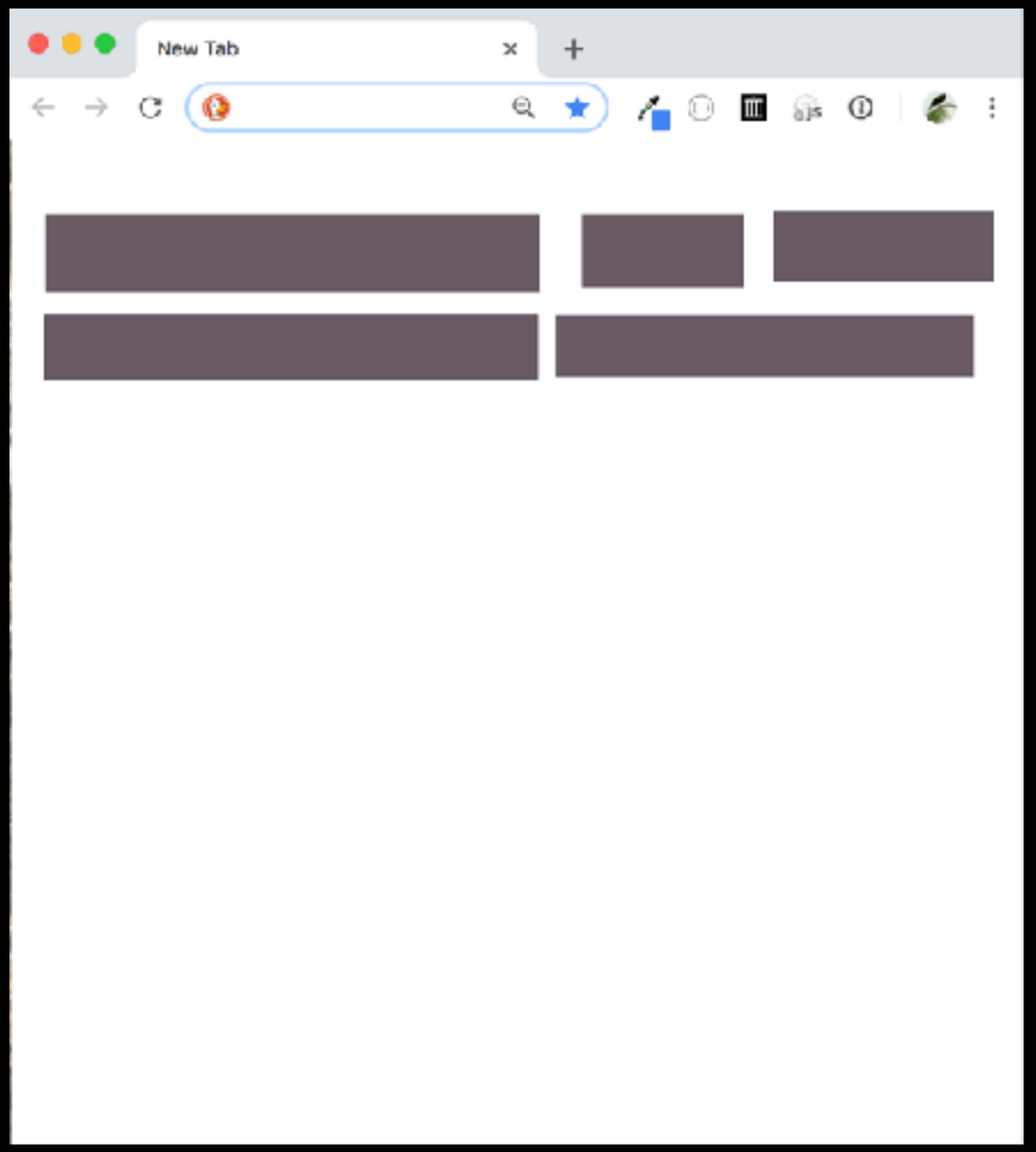
flexbox



Block layout

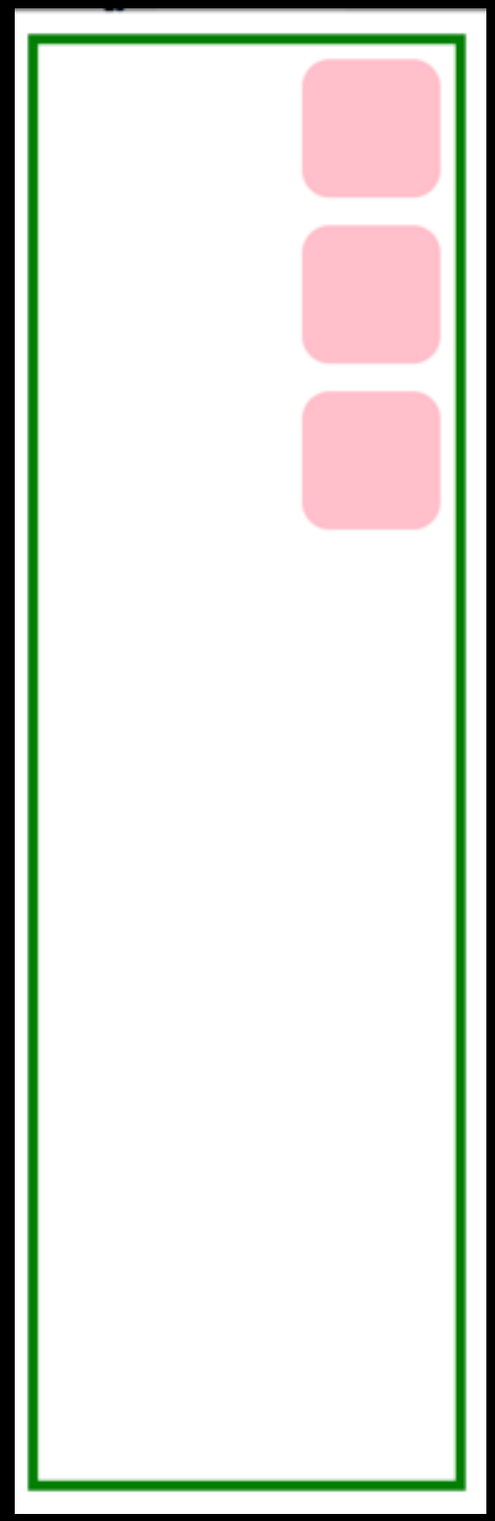
Laying out large sections of a page



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 - 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.

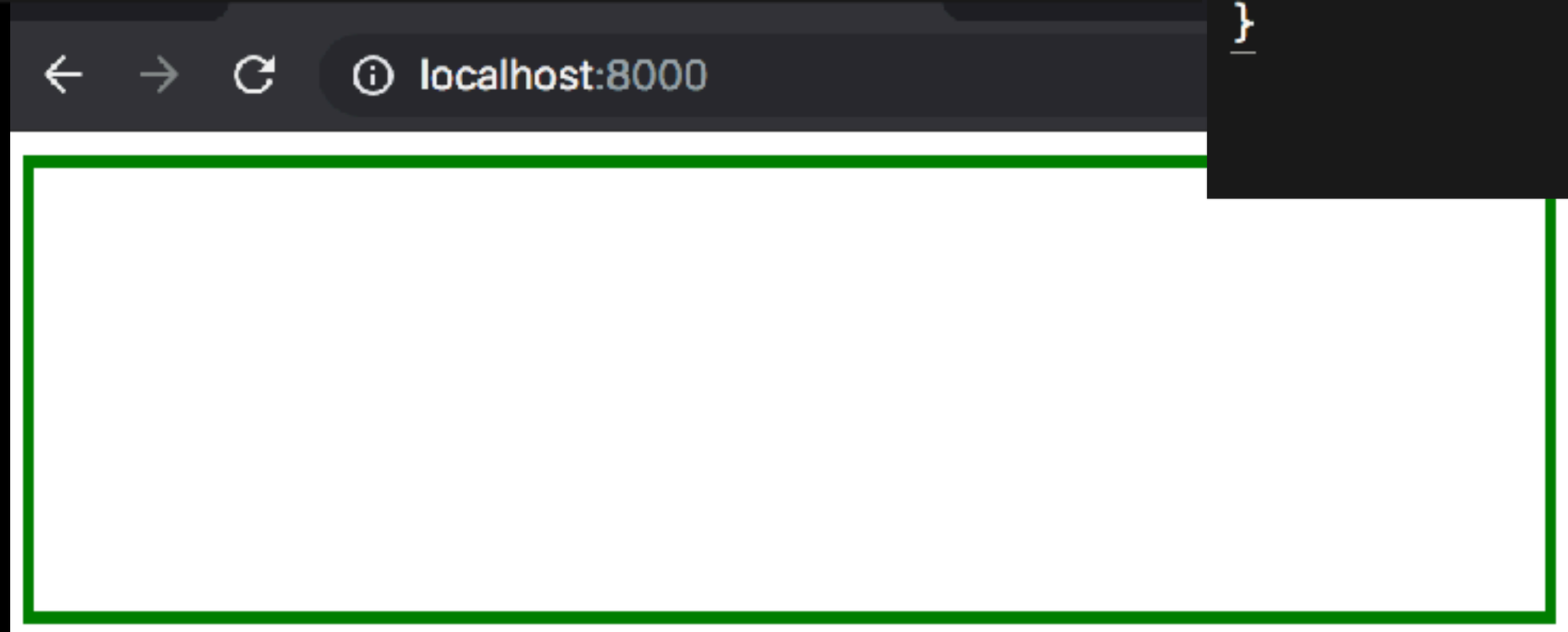
*width in the case of rows; height in the case of columns

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.

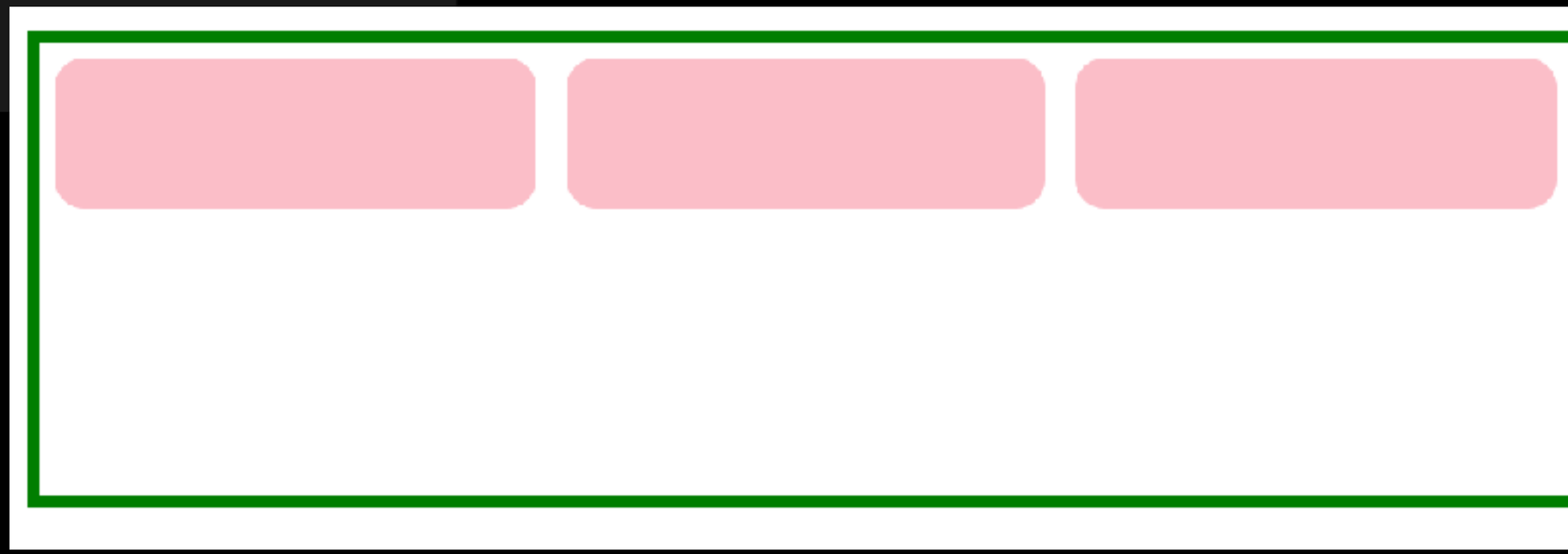
Flex items have **flex-shrink**: **1 by default**.

*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.

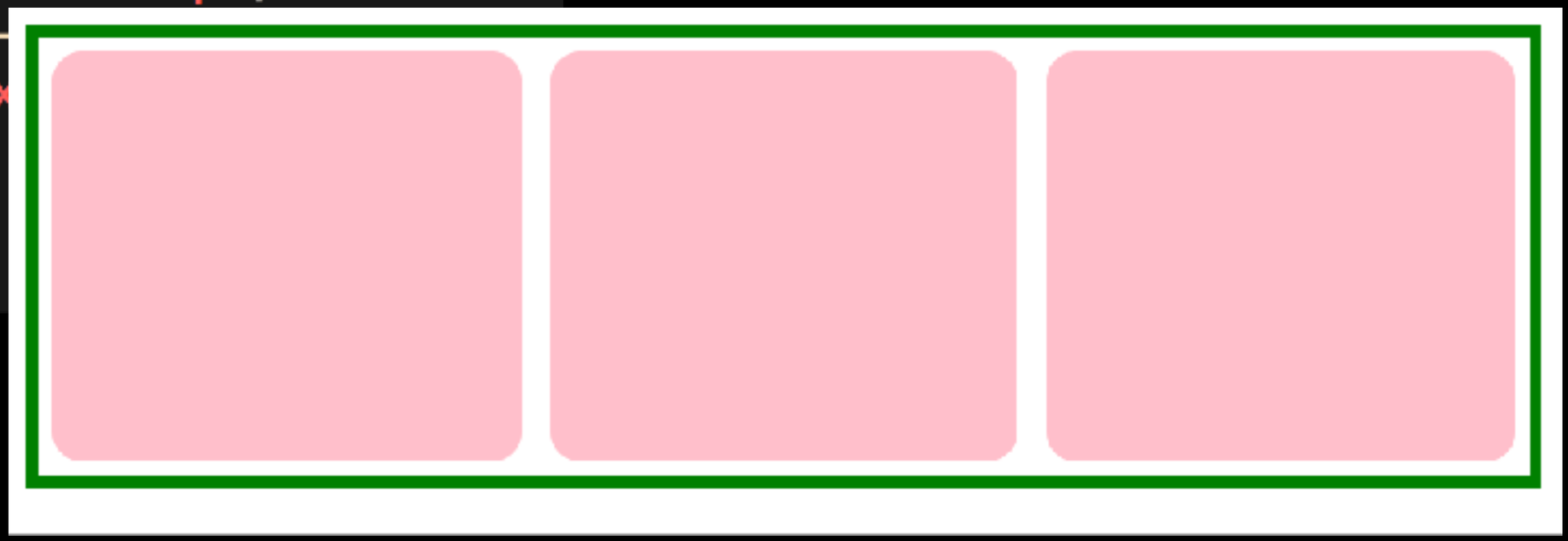
```
<div id="flexBox">
 <span class="flexThing"></span>
 <div class="flexThing"></div>
 <span class="flexThing"></span>
</div>
```

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

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

flex-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
}
```

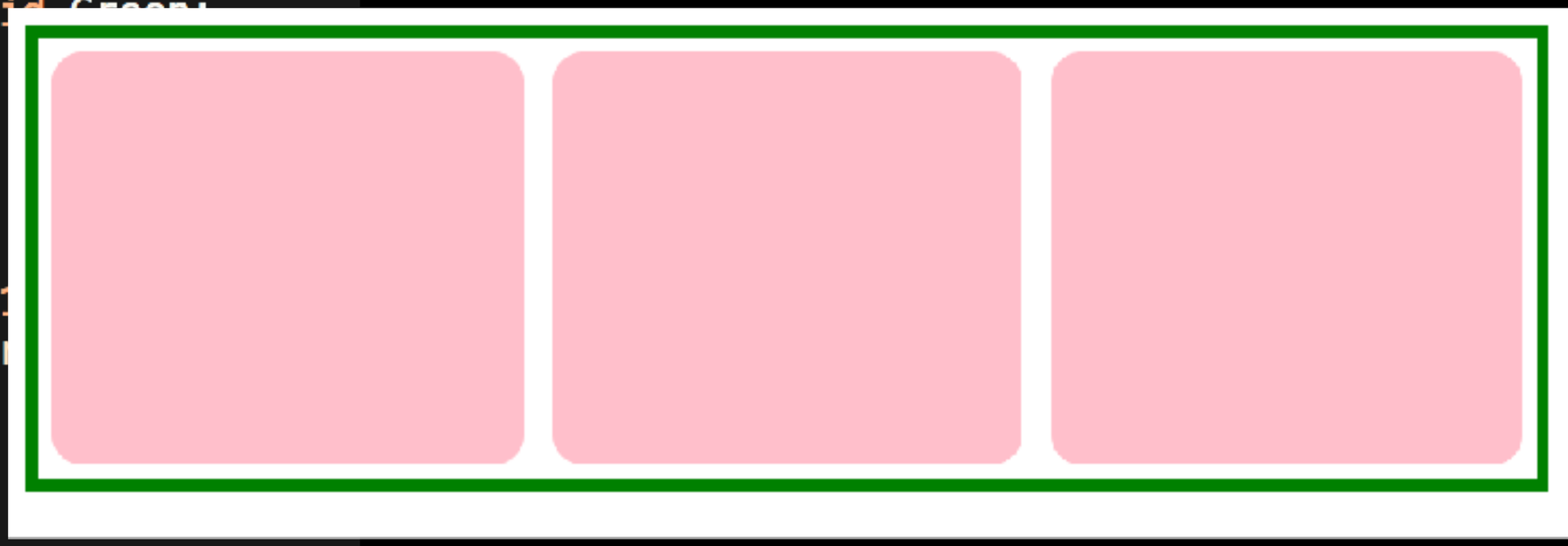


flex item height?**

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:
    background-colo
    margin: 5px;
}
```



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

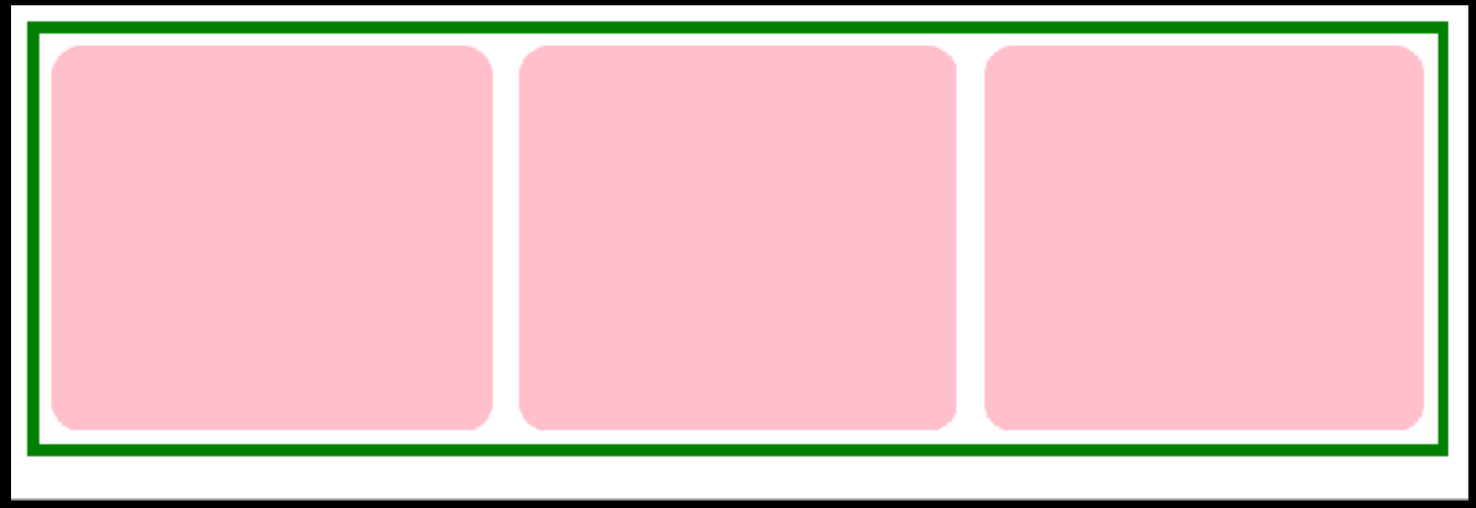
**height in the case of rows; width in the case of columns

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;
}
```



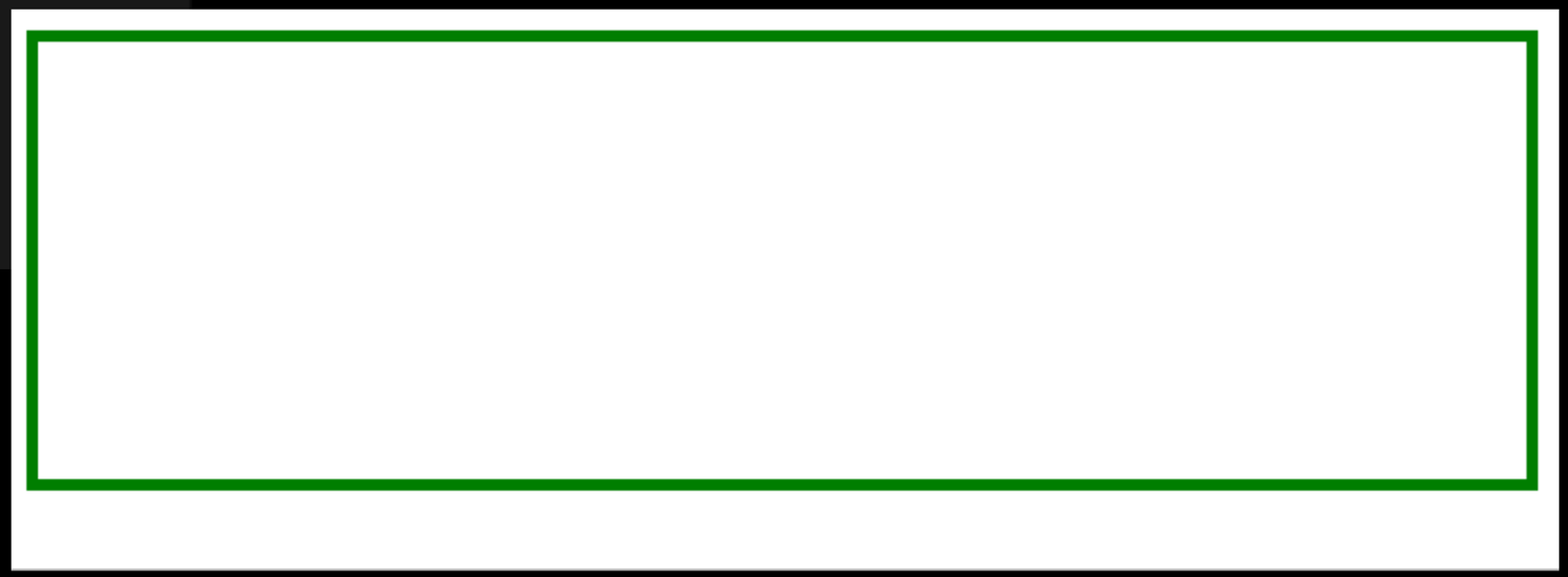
*veritcally in the case of rows; horizontally in the case of columns

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 - grids

Flexbox & CSS Grid

"The basic difference between CSS Grid Layout and CSS Flexbox Layout is that flexbox was designed for layout in one dimension - either a row or a column. Grid was designed for two-dimensional layout - rows, and columns at the same time. The two specifications share some common features, however, and if you have already learned how to use flexbox, the similarities should help you get to grips with Grid."

MDN

The flex-direction property defines in which direction the container wants to stack the flex items - either flex-direction: row or flex-direction: column. However, by using flex-wrap property. Read all about **CSS Flexbox @ W3**.

CSS Grid

A grid is an intersecting set of horizontal and vertical lines - one set defining columns and the other rows. Elements can be placed onto the grid, respecting these column and row lines.

How Grid Layout Works

The process for using the CSS Grid Layout Module is fundamentally simple:

+ Use the display property to turn an element into a grid container. The element's children automatically become grid items.
+ Set up the columns and rows for the grid. You can set them up explicitly and/or provide directions for how rows and columns should get created on the fly (the css grid is very flexible).
+ Assign each grid item to an area on the grid. If you don't assign them explicitly, they flow into the cells sequentially.

The element that has the display: **grid property** applied to it becomes the grid container and defines the context for grid formatting. All of its direct child elements automatically become grid items that end up positioned in the grid. You can define an explicit grid with grid layout but the specification also deals with the content added outside of a declared grid, which adds additional rows and columns when needed. Features such as adding "as many columns that will fit into a container" are included.

Grid line

The horizontal and vertical dividing lines of the grid are called grid lines.

Grid cell

The smallest unit of a grid is a grid cell, which is bordered by four adjacent grid lines with no grid lines running through it.

Grid area

A grid area is a rectangular area made up of one or more adjacent grid cells.

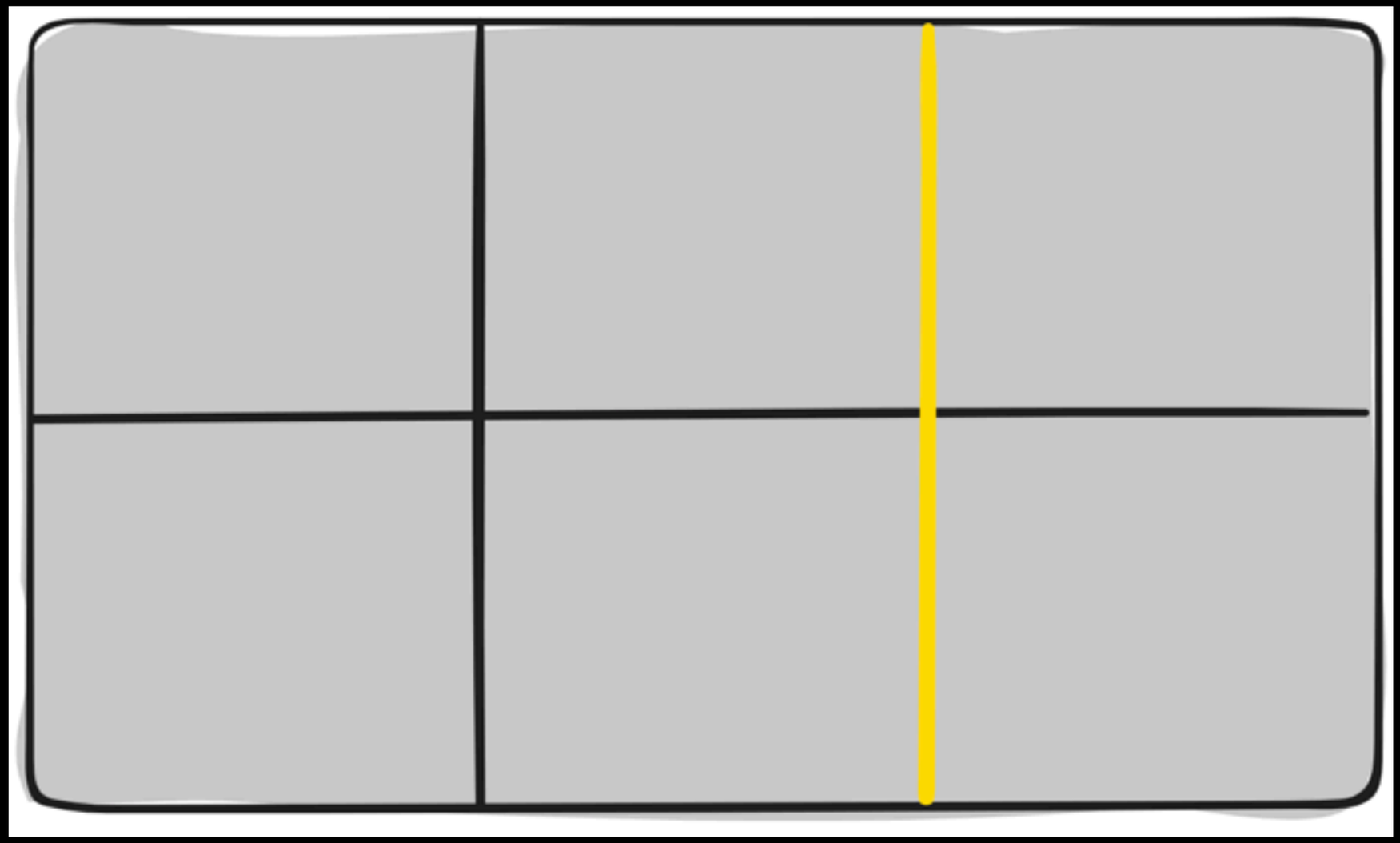
Grid track

The space between two adjacent grid lines is a grid track, which is a generic name for a grid column or a grid row. Grid columns are said to go along the block axis, which is vertical (as block elements are stacked) for languages written horizontally. Grid rows follow the inline (horizontal) axis.

The structure established for the grid is independent from the number of grid items in the container. You could place 4 grid items in a grid with 12 cells, leaving 8 of the cells as 'whitespace.' That's the flexibility of grids. You can also set up a grid with fewer cells than grid items, and the browser adds cells to the grid to accommodate them.

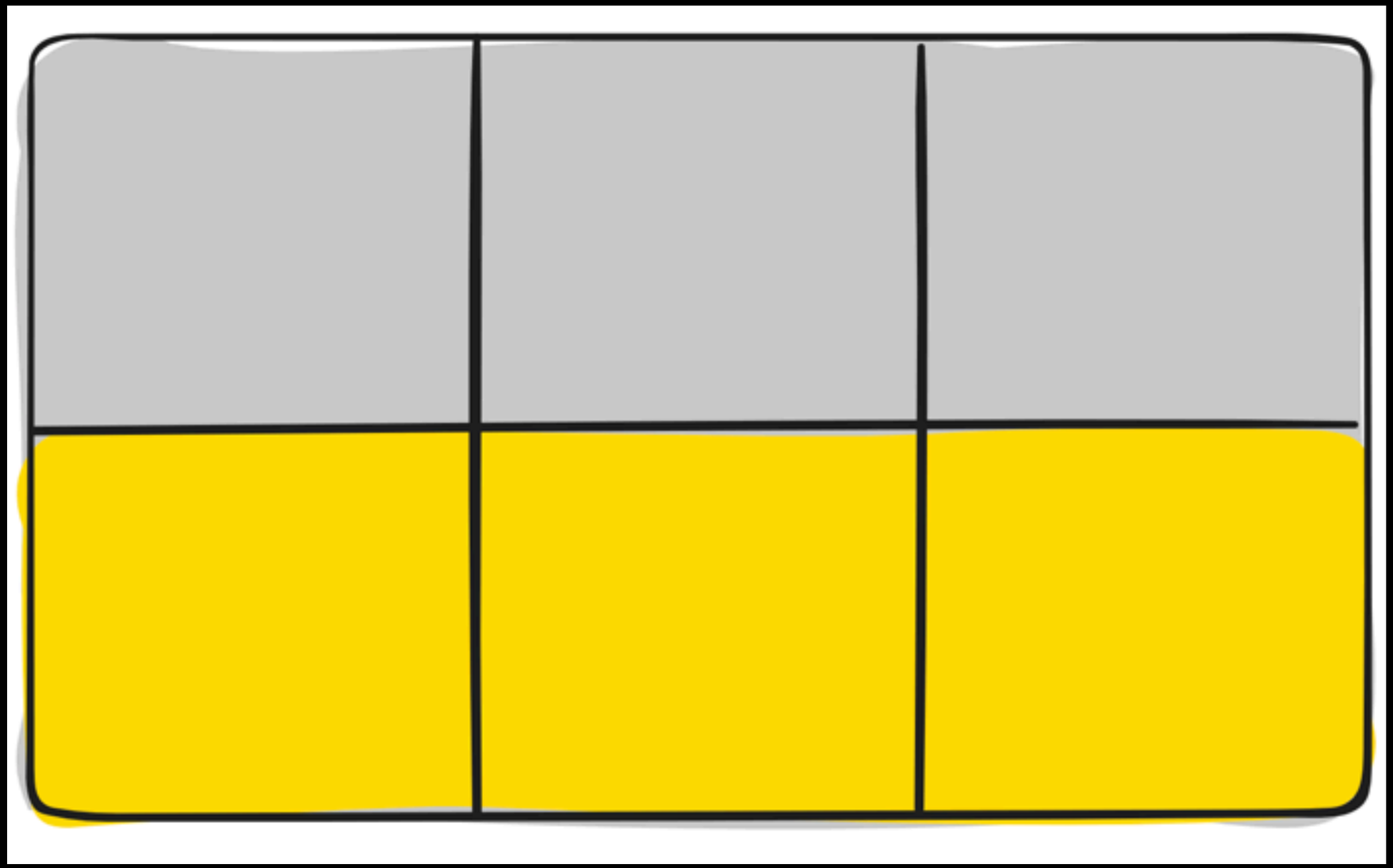
Grid line

The horizontal and vertical dividing lines of the grid are called grid lines. They can be either vertical ("column grid lines") or horizontal ("row grid lines") and reside on either side of a row or column. Here the yellow line is an example of a column grid line.



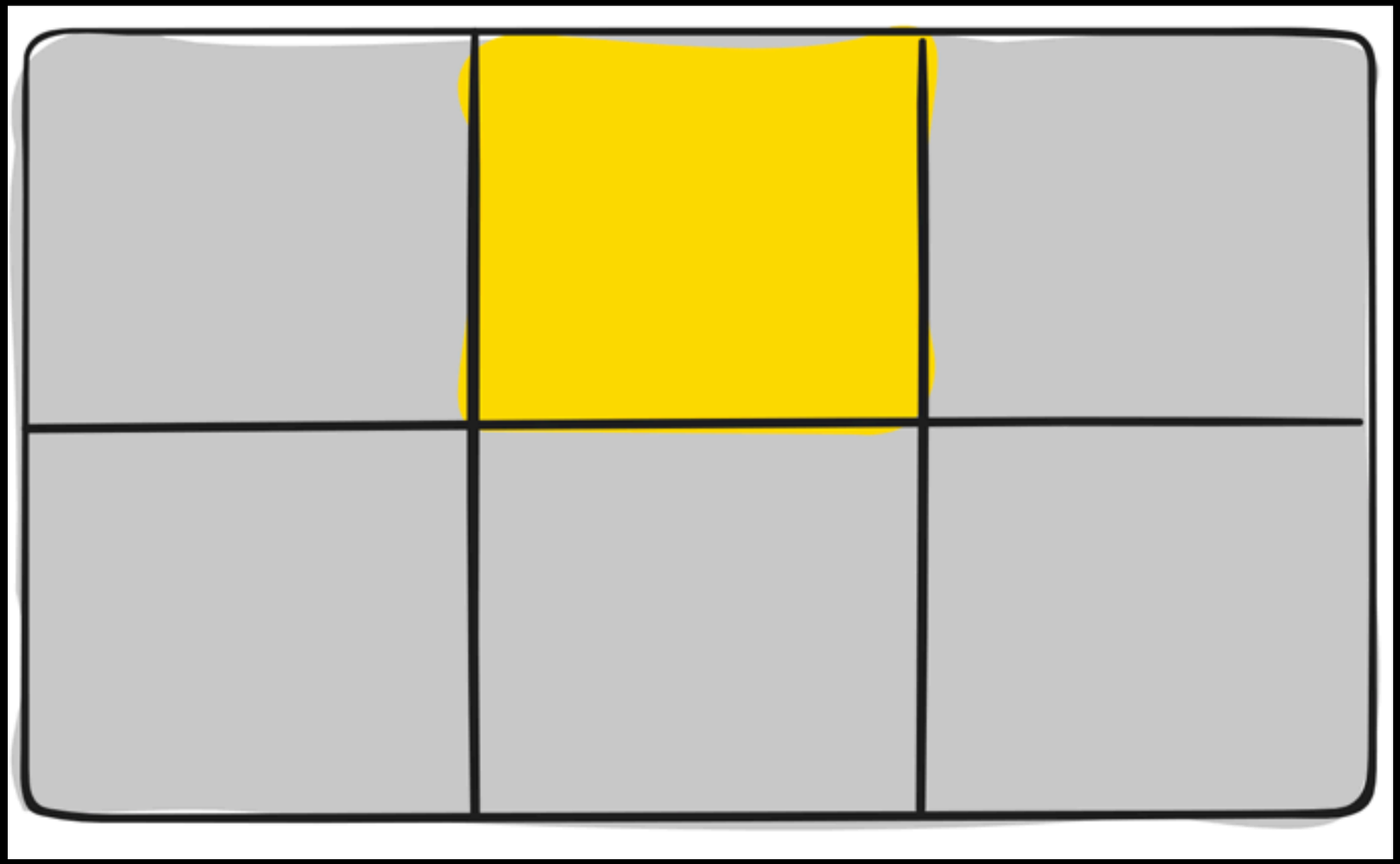
Grid track

The space between two adjacent grid lines is a grid track, which is a generic name for a grid column or a grid row. Grid columns are said to go along the block axis, which is vertical (as block elements are stacked) for languages written horizontally. Grid rows follow the inline (horizontal) axis. Here's the grid track between the second and third row grid lines.



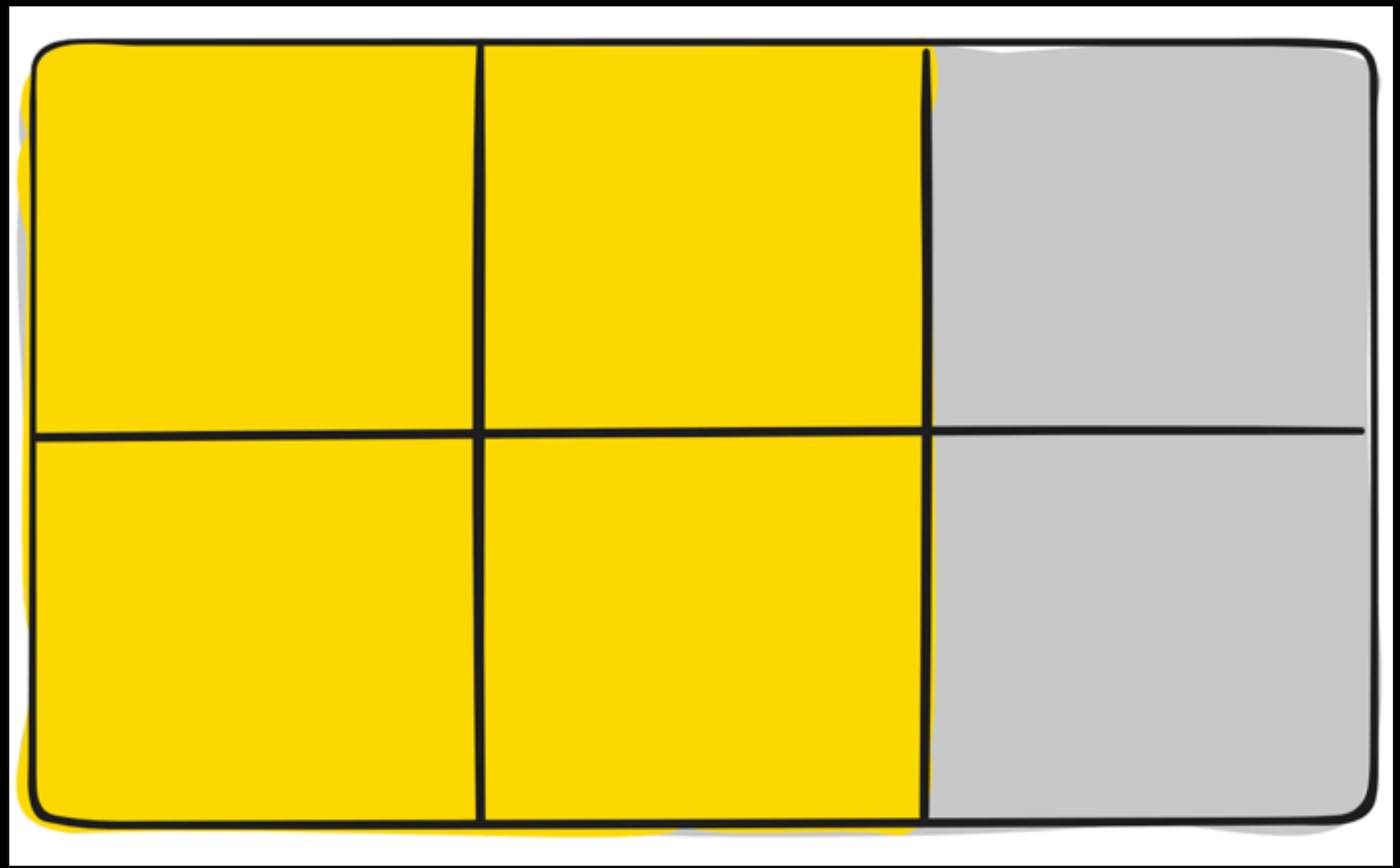
Grid cell

The smallest unit of a grid is a grid cell, which is bordered by four adjacent grid lines with no grid lines running through it. It's a single "unit" of the grid. Here's the grid cell between row grid lines 1 and 2, and column grid lines 2 and 3.



Grid area

A grid area is a rectangular area made up of one or more adjacent grid cells. A grid area may be comprised of any number of grid cells. Here's the grid area between row grid lines 1 and 3, and column grid lines 1 and 3.



Grid Container Properties:

display

grid-template-columns

grid-template-rows

grid-template-areas

grid-template

grid-column-gap

grid-row-gap

grid-gap

justify-items

align-items

place-items

justify-content

align-content

place-content

grid-auto-columns

grid-auto-rows

grid-auto-flow

grid

CSS Tricks w/ links!

Grid Item Properites

grid-column-start

grid-column-end

grid-row-start

grid-row-end

grid-column

grid-row

grid-area

justify-self

align-self

place-self

CSS Grid Functions

repeat ( )

minmax ( )

fit-content ( )

Fr unit

flexible length

Pair programming

In groups of 2:

Ideate + come up with a concept. Choose a theme or subject such as interiors or sad animated gifs...

Create two different layouts using CSS Grid. It may be two separate web pages or both layouts in one page or even a grid nested in a grid. It also must include 1 flexbox, feed back for the user (relative links, hover changes, etc).

Include text, image and/or video in each page. The layout themselves are entirely up to you - the goal is to use css grid + flex in different formats + learn from one another.

For Monday: Get your site running on each of yr servers. Do NOT post yr project site to the class wiki. Be prepared to present your finished sites during critique in class on Monday.

In groups of 2:

With your partner - come up with a concept + design strategy. Thinking about + testing the grid + flex properties - create a wire frame for your layouts including text + images. You can do this by hand or using software. Be prepared to present + discuss your site idea in 40 minutes.

You will then have the rest of class to begin coding your site. Use your peers (and me) ask questions, etc.