

the Box Model

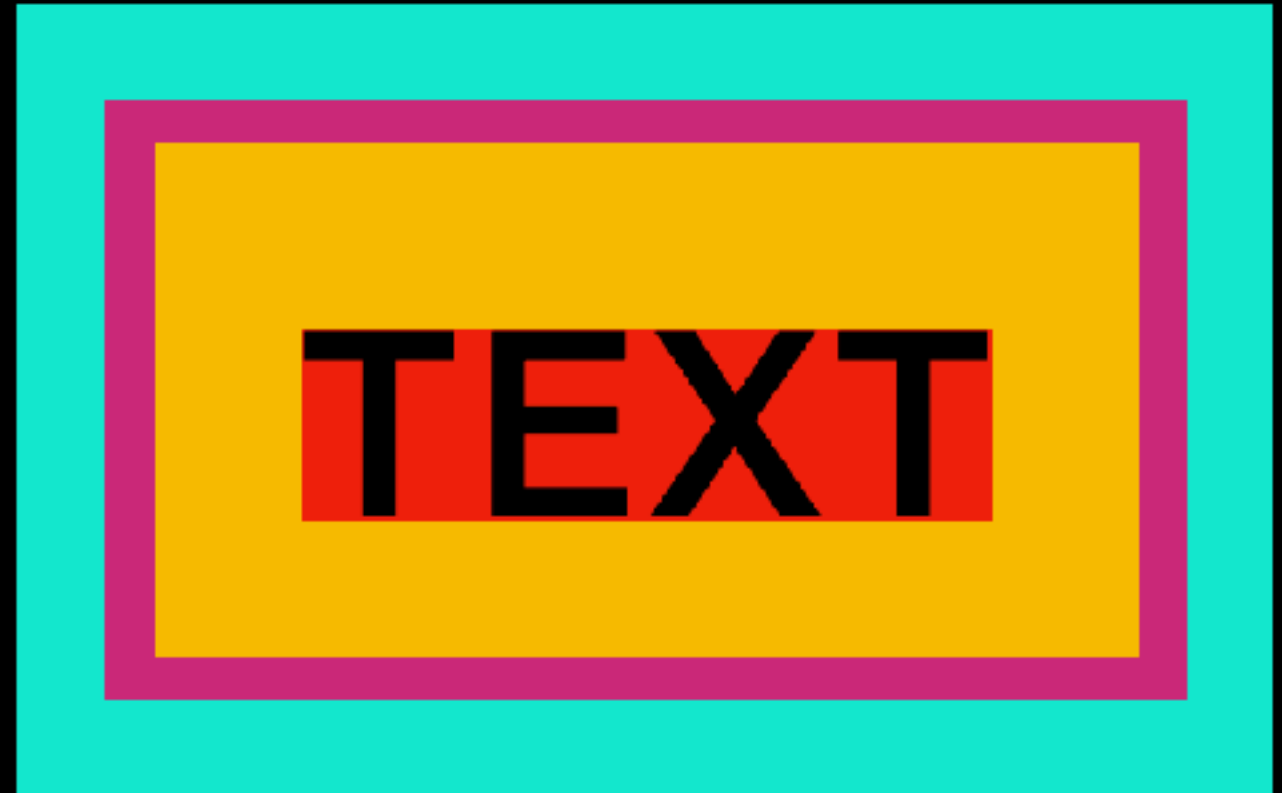
the Box Model

Border

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

Padding

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



Margin

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

Content

Display Property

`display: none;` — html elements default **visible**
— override default html position

`display: inline;`
`display: block;`

— responsive way to deal with positioning

`display: flex;`
`display: grid;`

Overriding Default Display

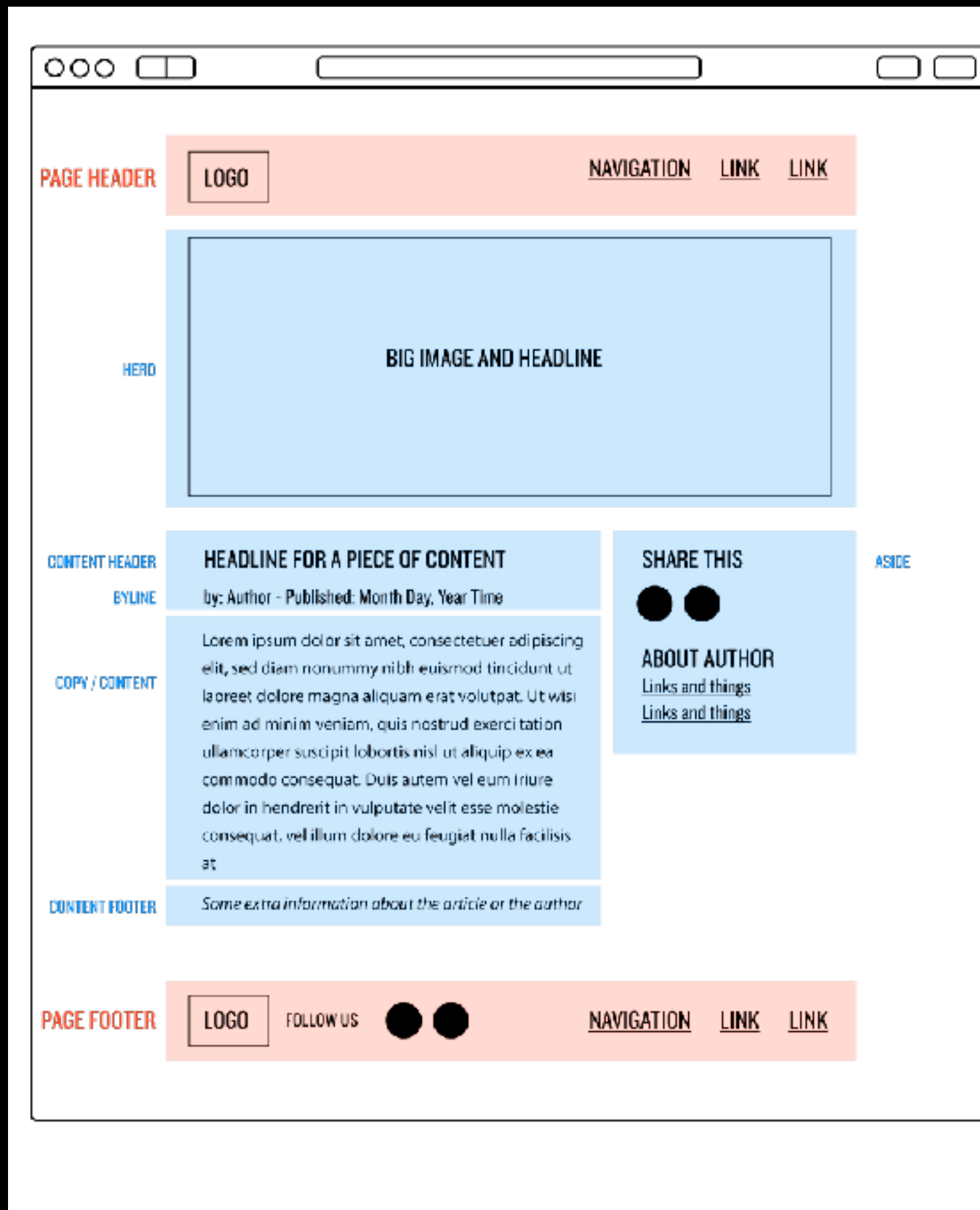
Changing an inline element to a block element, or vice versa, can be useful for making the page look a specific way, and still follow the web standards.

```
li {  
    display: inline;  
}
```

```
span {  
    display: block;  
}
```

Note: Setting the display property of an element only changes how the element is displayed, NOT what kind of element it is. So, an inline element with display: block; is not allowed to have other block elements inside it.

css layout



— Learn Enough CSS + Layout

So basically up until now I've instructed to do things a particular way. Bc w/ html, git, unix, etc there is only one way to do something (or a piece of software over the process). W/ CSS - there is no "right" answer. When designing websites many solutions to yr problem will exist - which means subjective judgment is the rule rather than the exception.

— Learn Enough CSS + Layout

You have to get used to the idea that no site is going to be exactly the same when viewed by different people. You'll learn to design (or implement other people's designs) in a way that allows room for CSS's inherent ambiguity. Unlike the tightly constrained world of print design, getting things to look exactly the same in every browser and on every operating system is just something you have to give up worrying about.

— Learn Enough CSS + Layout

Typeface terminology

Serif

A serif is a small decorative flourish on the end of the strokes that make up letters and symbols.

Sans Serif

Sans Serif fonts do not have any flourishes at the end of strokes.

Monospaced

Monospaced fonts, letters, and characters each occupy the same amount of horizontal space.

Georgia
Times
Times New Roman

Arial
Verdana
Helvetica

Courier
Courier New

Default Web Fonts

Verdana

Arial

Arial Narrow

Arial Black

Helvetica

Century Gothic

Courier

Courier New

COPPERPLATE GOTHIC

Times

Times New Roman

Georgia

Geneva

Gill Sans

Tahoma

Trebuchet

Comic Sans

Impact

Palatino Linotype

Book Antiqua

Lucida Console

Lucida Sans Unicode

Serif

Sans-Serif

Font stack

It's important to understand that the browser will only display font if it's installed on user's computer.

Font stack - a collection of more than one typeface in an order of preference to be displayed in the browser if some of the typefaces are not found.

```
{  
  font-family: Georgia, Courier, serif;  
}
```

Font

Padding is the space btw the border + the content.

Some Properties:

font-family

color

font-size

line-height

text-align

Font

Padding is the space btw the border + the content.

text-decoration

underline, strike thru or none (eg to unset underline on hyperlinks)

text-transform

change font **case** (eg uppercase, lower, capitalize, none)

font-style

set to italic or normal

font-weight

set to bold or normal

letter-spacing

controls the space btw letters

font-family property sets the font in your CSS

Presented as a hierarchy of choices (1st choice, 2nd choice, 3rd choice) so it's good to have a fallback for older browsers that can't render

```
body {
```

```
    font-family: Georgia, Courier, serif;
```

```
}
```

```
h1, h2, h3 {
```

```
    font-family: Arial, Verdana, sans-serif;
```

```
}
```

Custom web fonts: Google Fonts

Add link in **<head>** of HTML

```
<link href="https://fonts.googleapis.com/css?
family=Roboto" rel="stylesheet">
```

Use with font-family property in CSS

```
font-family: 'Roboto', sans-serif;
```

css units of
measurement:

Absolute length units

The following are all **absolute** length units — they are not relative to anything else, and are generally considered to always be the same size.

Unit	Name	Equivalent to
cm	Centimeters	1cm = 38px = 25/64in
mm	Millimeters	1mm = 1/10th of 1cm
Q	Quarter-millimeters	1Q = 1/40th of 1cm
in	Inches	1in = 2.54cm = 96px
pc	Picas	1pc = 1/6th of 1in
pt	Points	1pt = 1/72th of 1in
px	Pixels	1px = 1/96th of 1in

https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/Values_and_units

VIEWPORT WIDTH // VIEWPORT HEIGHT

- 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

```
div {  
  width: 10vw;  
  height: 10vw;  
}
```

responsive text

The text size can be set with a "vw" unit, which means the "viewport width".

That way the text size will follow the size of the browser window.

```
div {  
  font-size:10vw;  
}
```

Media Queries

the @media rule tells the browser to include a block of CSS properties only if a certain condition is true.

So this:

```
@media only screen and (max-width: 500px) {  
  body {  
    background-color: light blue;  
  }  
}
```

Translates to:

```
if (the maximum width of the web page is 500 pixels) {  
  then do this stuff  
}
```

Relative length units

Relative length units are relative to something else, perhaps the size of the parent element's font, or the size of the viewport. The benefit of using relative units is that with some careful planning you can make it so the size of text or other element scales relative to everything else on the page. Some of the most useful units for web development are listed in the table below.

Unit	Relative to
em	Font size of the parent, in the case of typographical properties like <code>font-size</code> , and font size of the element itself, in the case of other properties like <code>width</code> .
ex	x-height of the element's font.
ch	The advance measure (width) of the glyph "0" of the element's font.
rem	Font size of the root element.
lh	Line height of the element.
vw	1% of the viewport's width.
vh	1% of the viewport's height.
vmin	1% of the viewport's smaller dimension.
vmax	1% of the viewport's larger dimension.

Media Queries

Breakpoint

```
/* For mobile phones: */
```

```
[class*="col-"] {  
    width: 100%;
```

```
}
```

```
@media only screen and (min-width: 768px) {
```

```
    /* For desktop: */
```

```
.col-1 {width: 8.33%;}
```

```
.col-2 {width: 16.66%;}
```

```
.col-3 {width: 25%;}
```

```
.col-4 {width: 33.33%;}
```

```
.col-5 {width: 41.66%;}
```

```
.col-6 {width: 50%;}
```

```
.col-7 {width: 58.33%;}
```

```
.col-8 {width: 66.66%;}
```

```
.col-9 {width: 75%;}
```

```
.col-10 {width: 83.33%;}
```

```
.col-11 {width: 91.66%;}
```

```
.col-12 {width: 100%;
```

```
}
```

add a **breakpoint** where certain parts of the design will behave differently on each side of the breakpoint

many examples: https://www.w3schools.com/Css/css_rwd_mediaqueries.asp

Mobile-first! (Images)



```
/* For width smaller than 400px: */  
body {  
    background-image: url('void_newspaper.jpg');  
}
```



```
/* For width 400px and larger: */  
@media only screen and (min-width: 400px) {  
    body {  
        background-image: url('void.jpg');  
    }  
}
```

responsive

“A pixel is not a pixel”
— Peter Paul Koch

“If the pixel density of the output device is very different from that of a typical computer display, the **user agent** should rescale pixel values. It is recommended that the pixel unit refer to the whole number of device pixels that best approximates the reference pixel. It is recommended that the reference pixel be the visual angle of one pixel on a device with a pixel density of 96dpi and a distance from the reader of an arm's length.” — [w3 consortium](#)

<!

- - Tells the browser to match the device's width for the viewport
- Sets an initial zoom value -->

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

<!-- Moving forward this line of code should be in EVERY web page you author. —>

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

```
1  <!DOCTYPE html>
2  <html>
3
4  <head>
5    <title> 🖥️ Web Dev Week 06</title>
6    <link href="https://fonts.googleapis.com/css?family=Spartan|Trade+Winds&display=swap" rel="stylesheet">
7    <meta charset="utf-8">
8    <meta name="viewport" content="width=device-width, initial-scale=1">
9    <link rel="stylesheet" type="text/css" href="theStyle.css">
10  </head>
11
12 <body>
```

`<!-- upcoming question - what is this? -->`

`<meta charset="utf-8">`

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



without



with

Metadata: `viewport`

The user's visible area of a web page

HTML5 introduced a method to let web designers take control over the viewport, through the `<meta>` tag.

Let's breakdown the `content` value:

- + **Values** are comma separated, letting you specify a list of values for `content`
- + The `width` value is set to `device-width`. This will cause the browser to render the page at the same width of the device's screen size.
- + `initial-scale` set to `1` indicates the "zoom" value if your web page when it is first loaded. `1` means "no zoom."

There are other values you can specify for the `content` list -

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

Metadata: `viewport`

There are other **values** you can specify for the **`content`** attribute -

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

500px

minimum-scale
maximum-scale
user-scalable

Flex - different rendering model

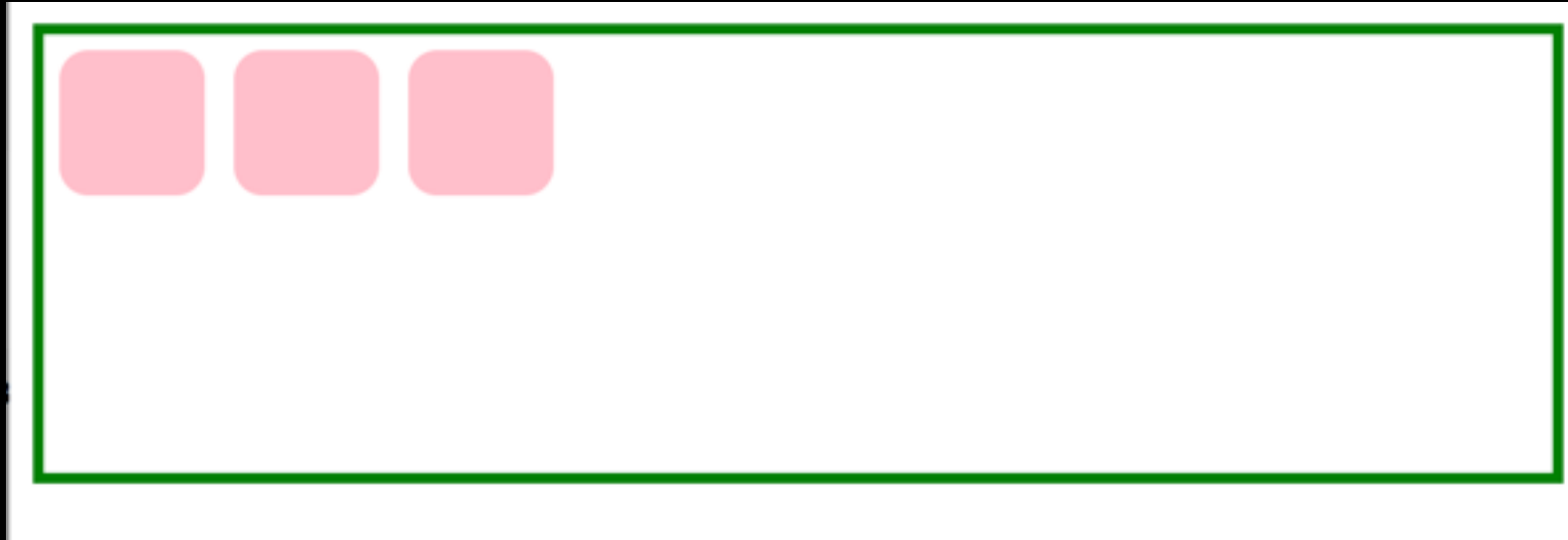
When you set a container to **display: flex**, the direct children in that container are **flex items** + follow a new set of rules.

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 display

Flex Basics



Flex layouts are composed of:

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

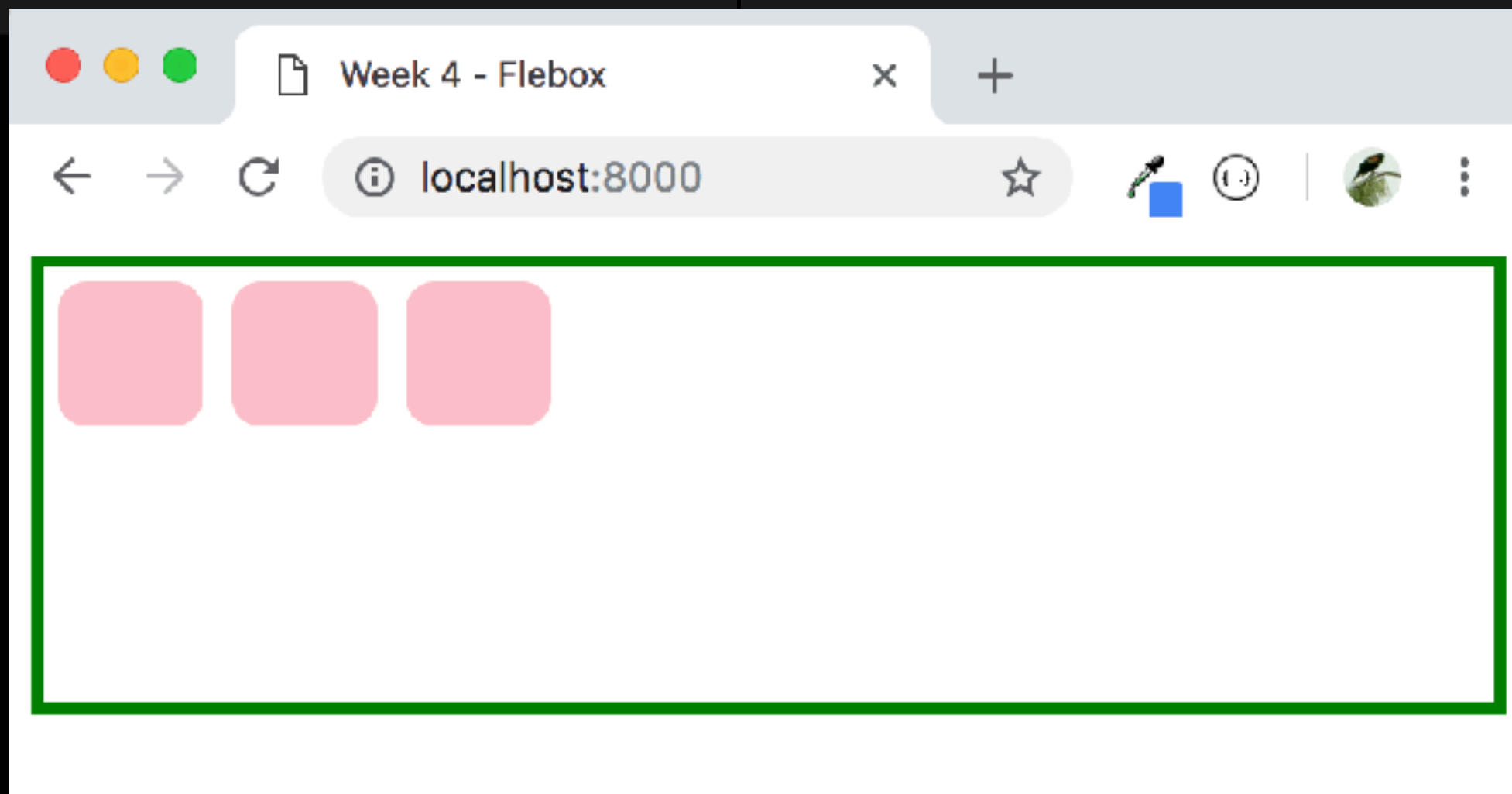
To make an element a flex container, change display:

- 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.

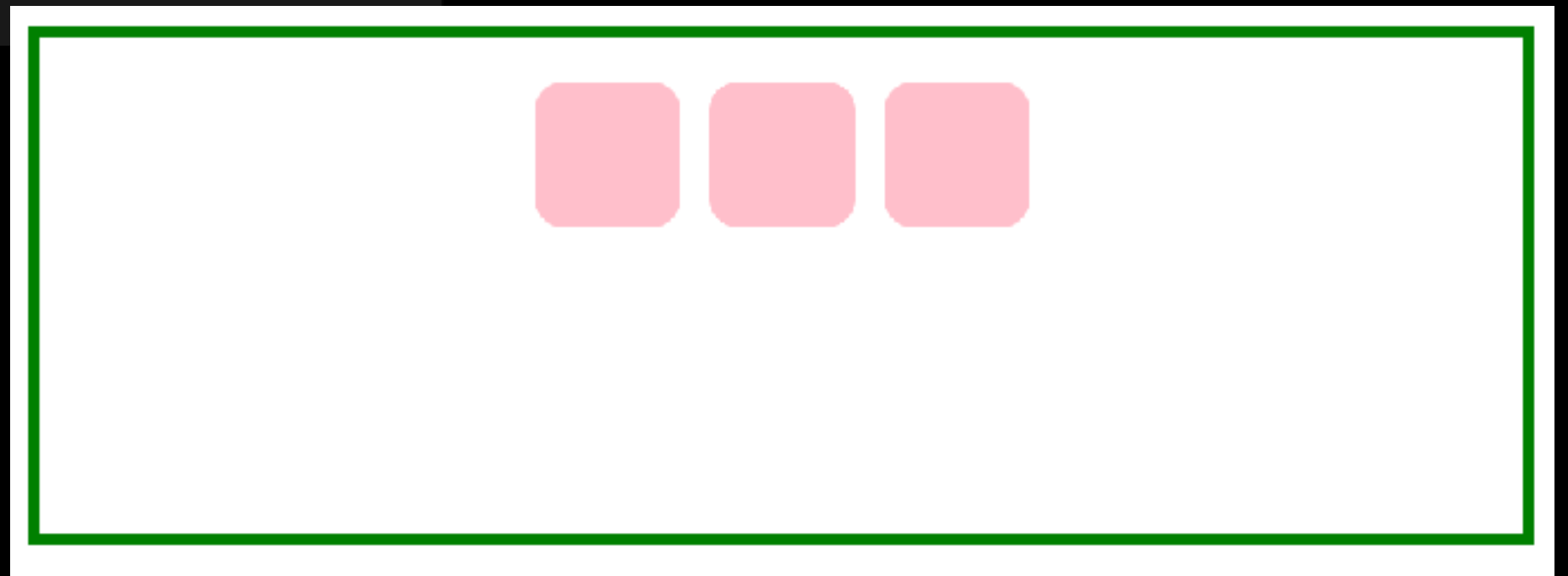
```
#flexBox {  
  display: flex;  
  justify-content: flex-end;  
  padding: 10px;  
  height: 150px;  
  border: 4px solid Green;  
}
```



Flex Basics: justify-content

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

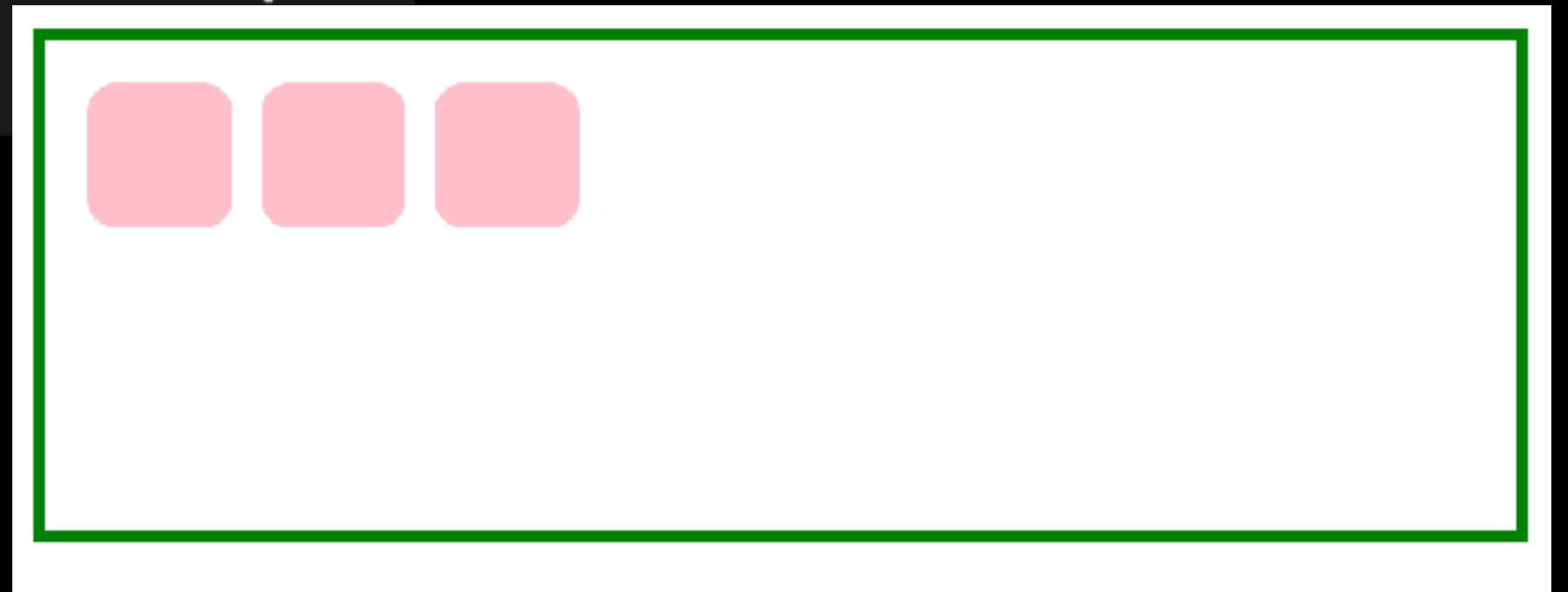
```
#flexBox {  
  display: flex;  
  justify-content: center;  
  padding: 10px;  
  height: 150px;  
  border: 4px solid Green;  
}
```



Flex Basics: align-items

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

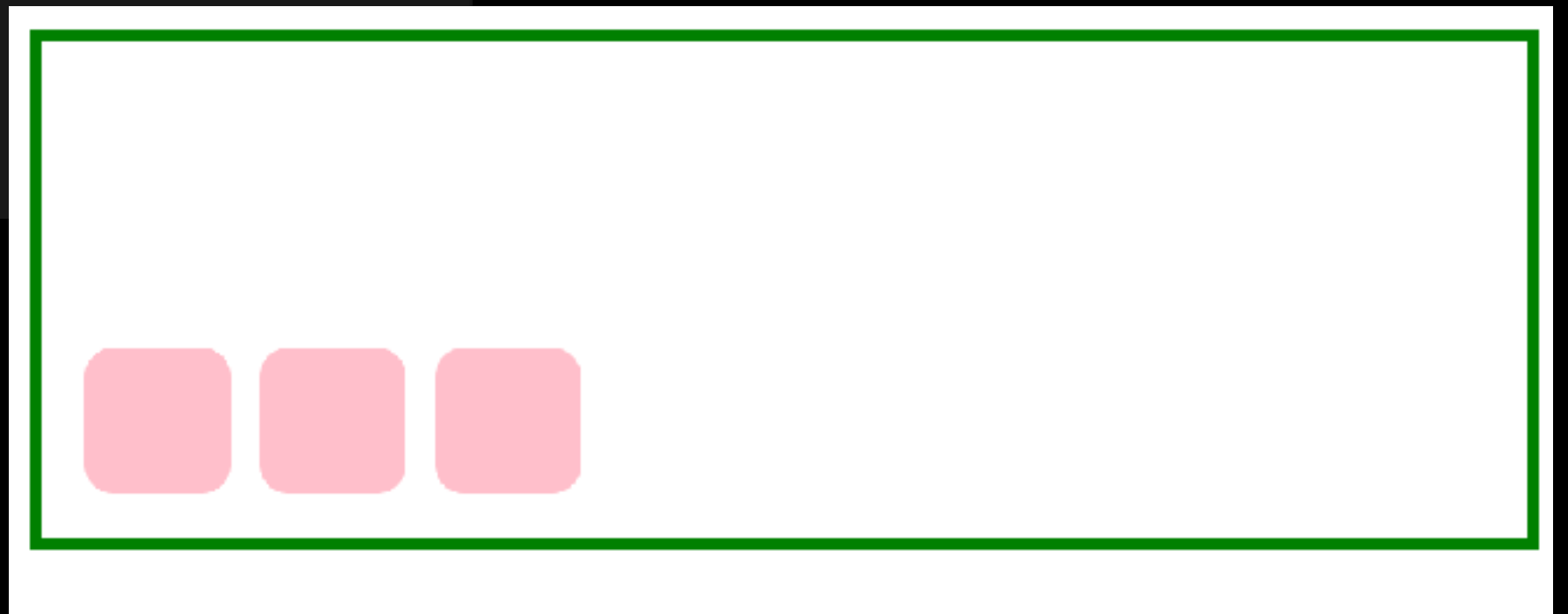
```
#flexBox {  
  display: flex;  
  align-items: flex-start;  
  padding: 10px;  
  height: 150px;  
  border: 4px solid Green;  
}
```



Flex Basics: align-items

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

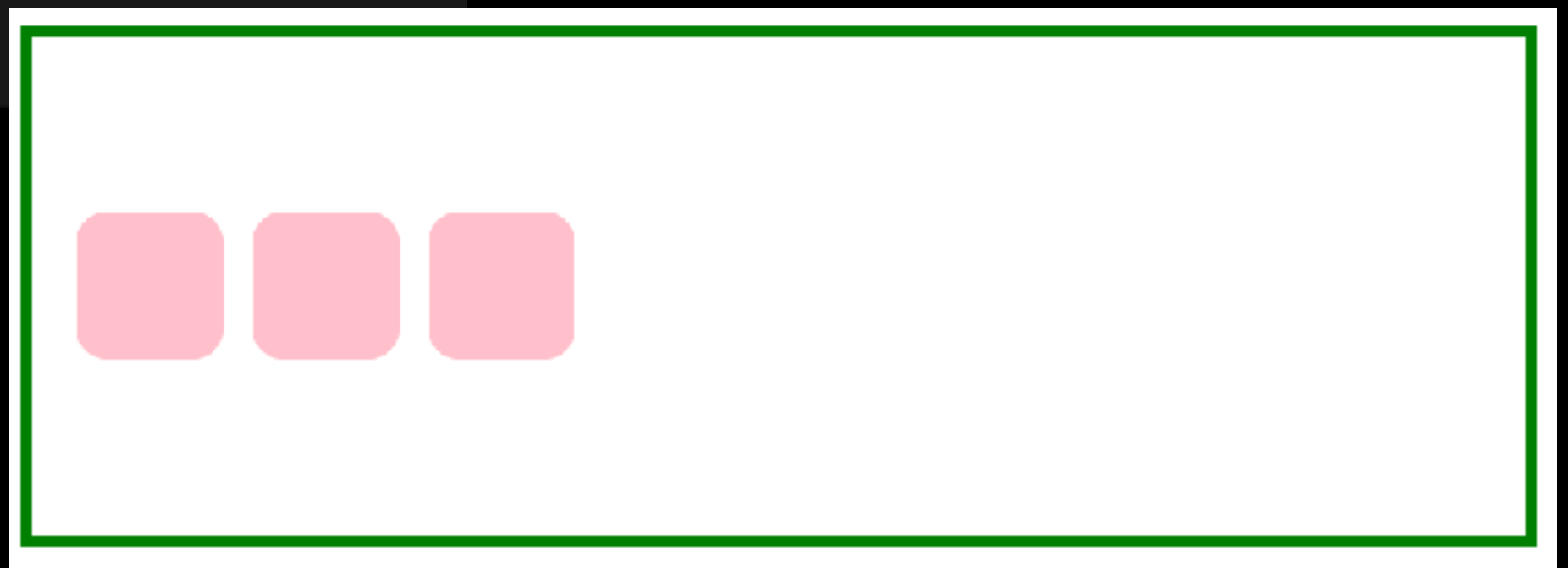
```
#flexBox {  
  display: flex;  
  align-items: flex-end;  
  padding: 10px;  
  height: 150px;  
  border: 4px solid  
}
```



Flex Basics: align-items

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

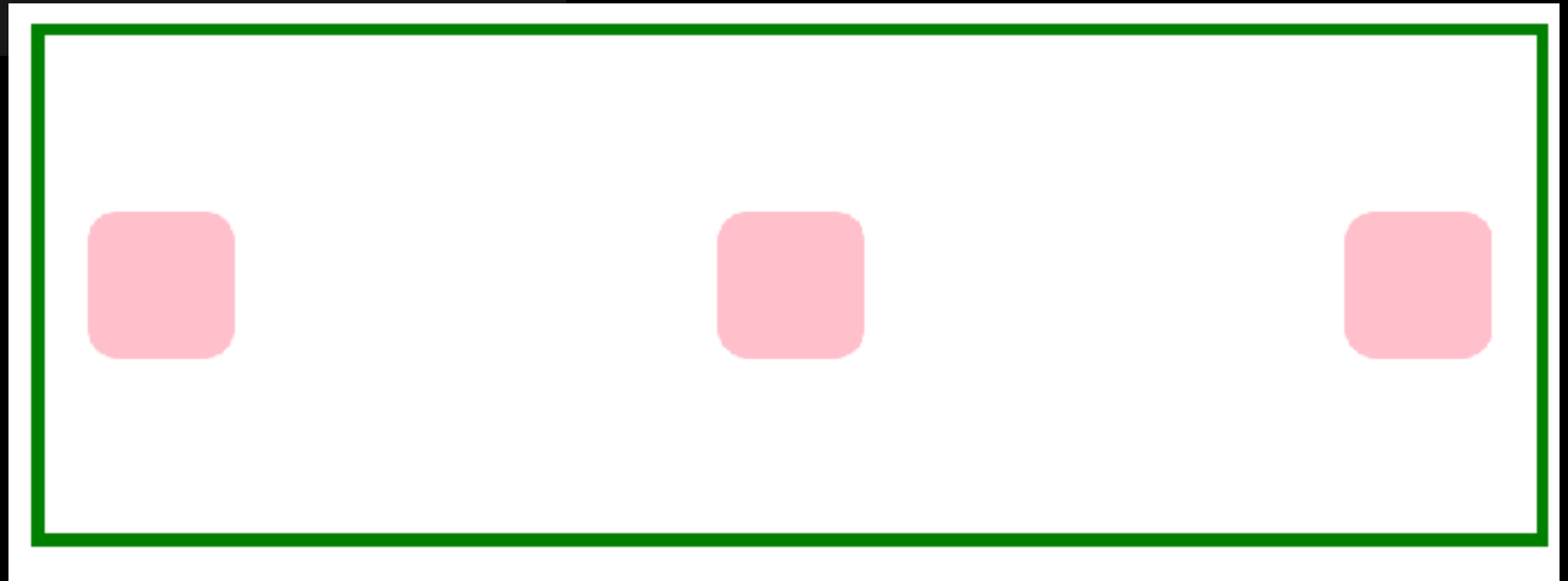
```
▼ #flexBox {  
  display: flex;  
  align-items: center;  
  padding: 10px;  
  height: 150px;  
  border: 4px solid Green;  
}
```



Flex Basics:

justify-content: space-between + space-around

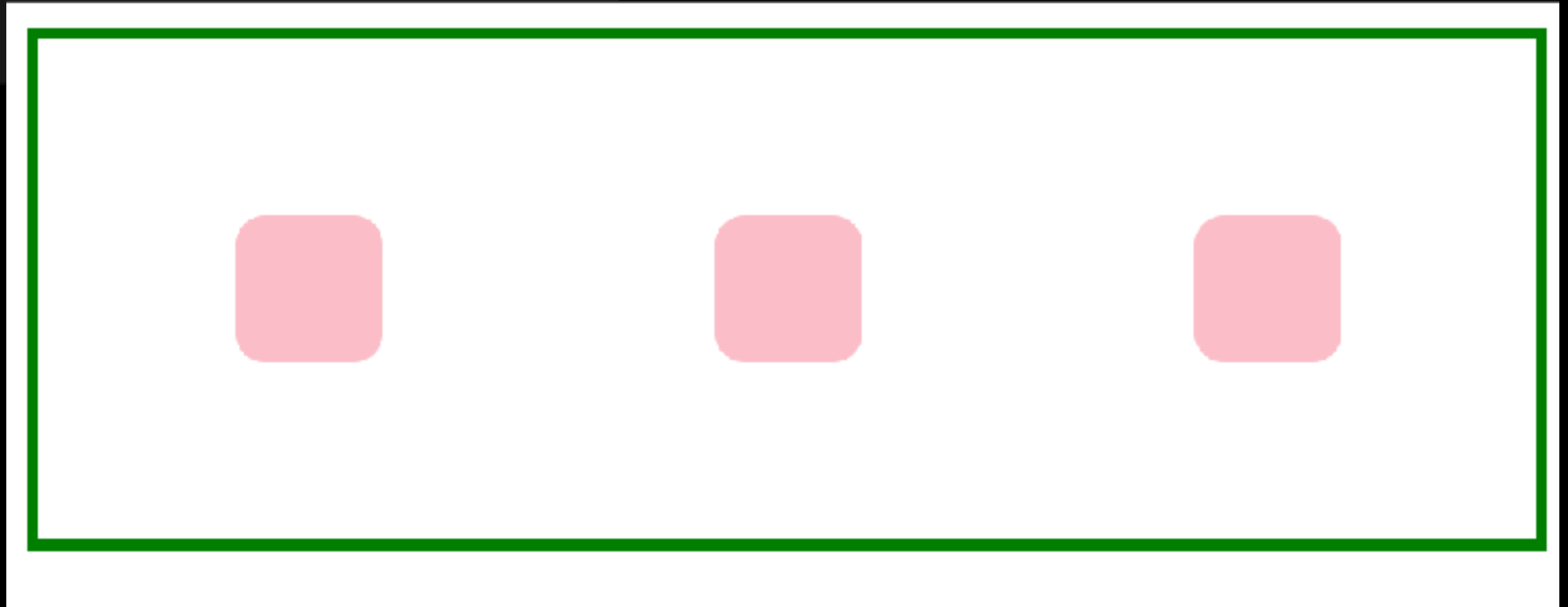
```
#flexBox {  
  display: flex;  
  justify-content: space-between;  
  align-items: center;  
  padding: 10px;  
  height: 150px;  
  border: 4px solid Green;  
}
```



Flex Basics:

`justify-content: 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

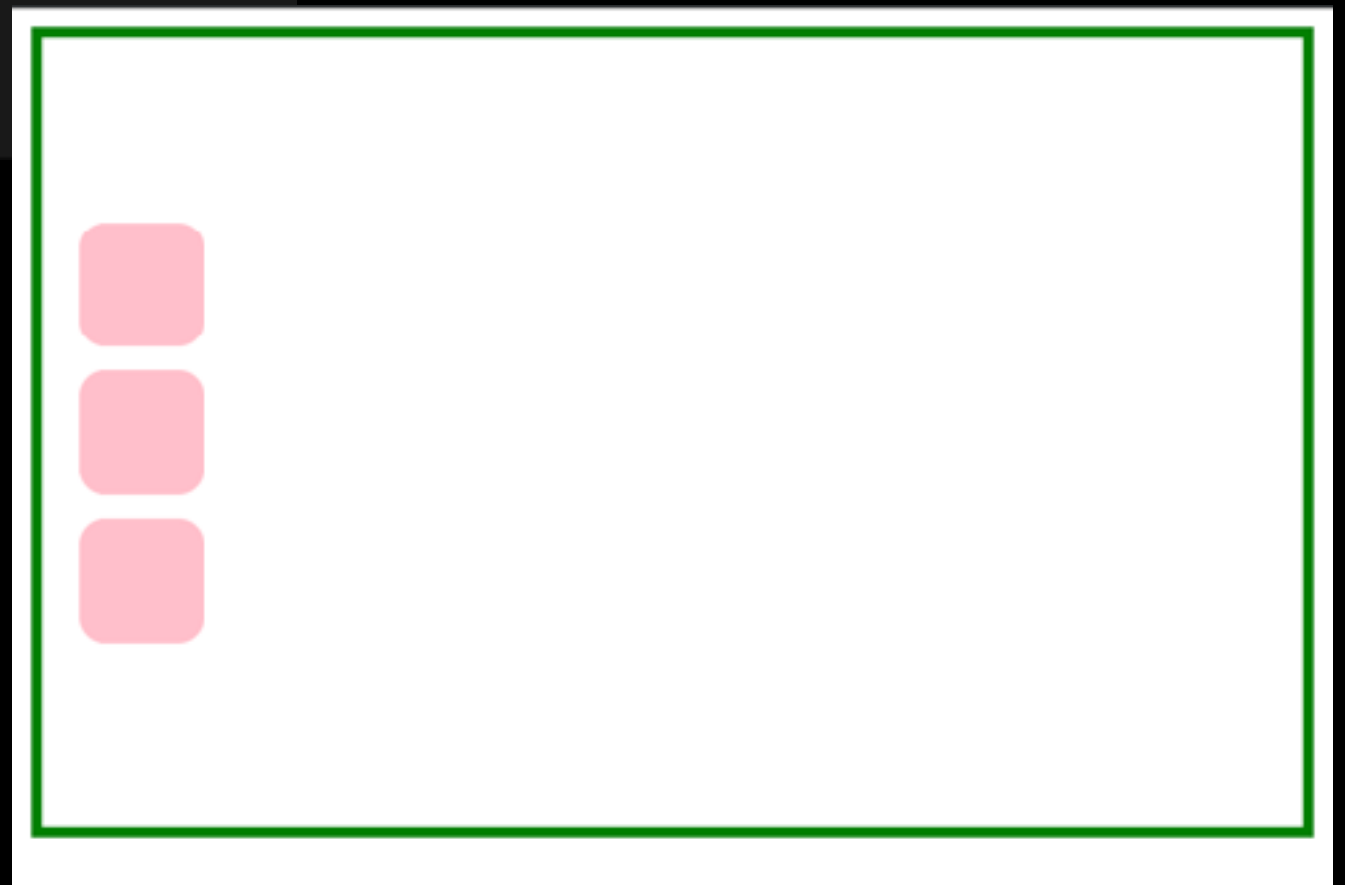
```
#flexBox {  
  display: flex;  
  flex-direction: column;  
  padding: 10px;  
  height: 150px;  
  border: 4px solid Green;  
}
```



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.

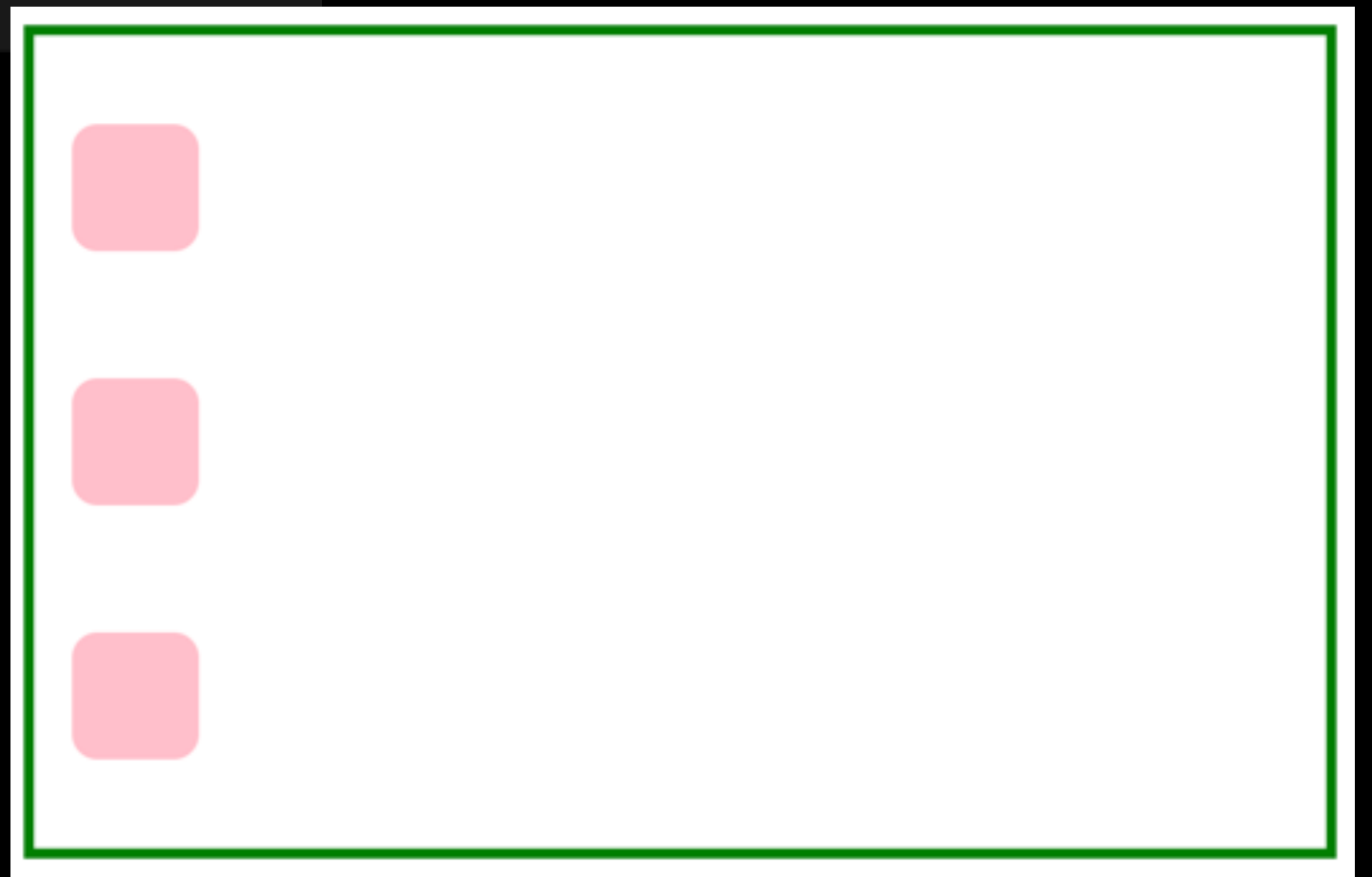


Flex Basics: flex-direction

```
▼ #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.

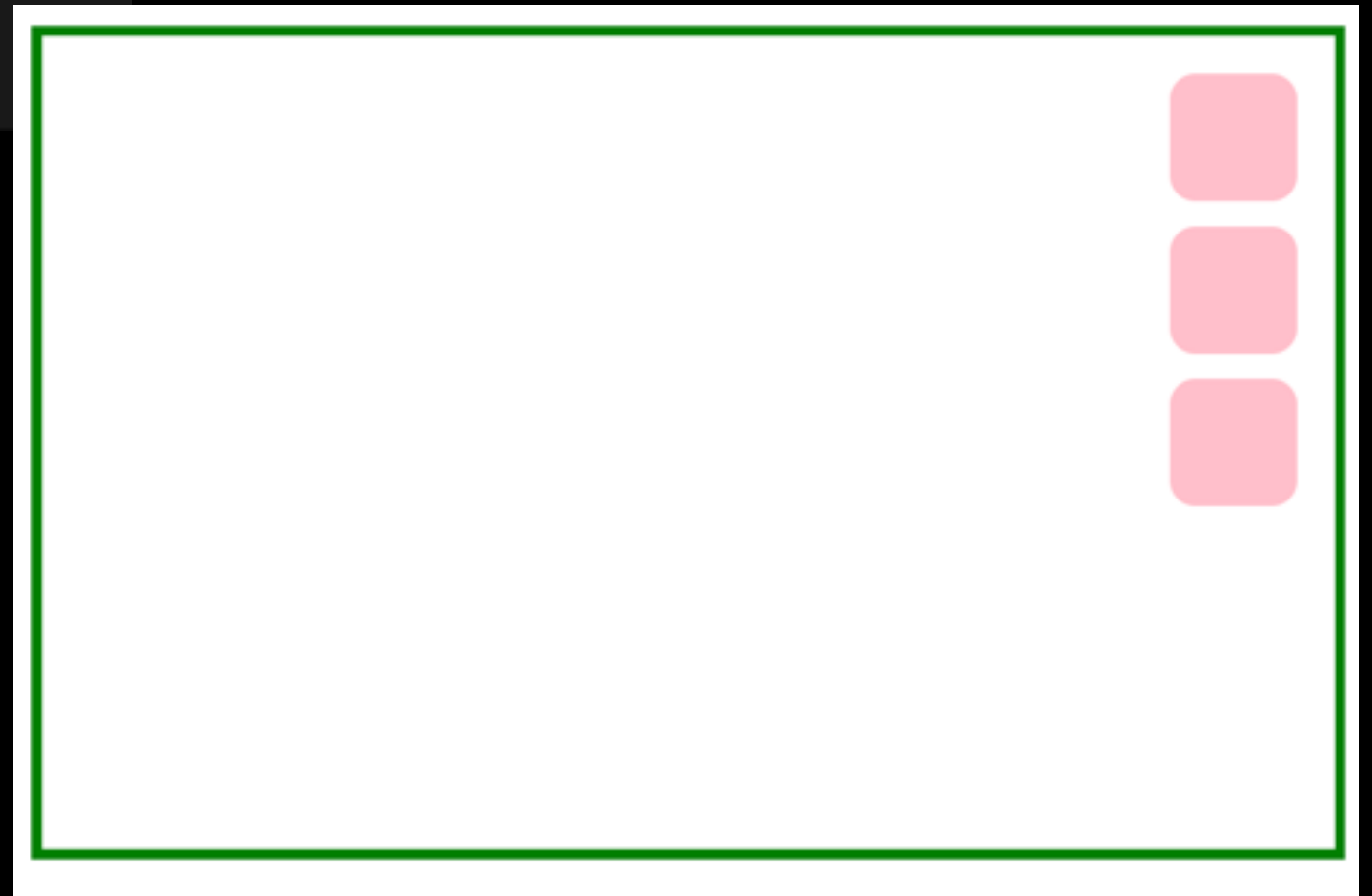


Flex Basics: flex-direction

```
▼ #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 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-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.

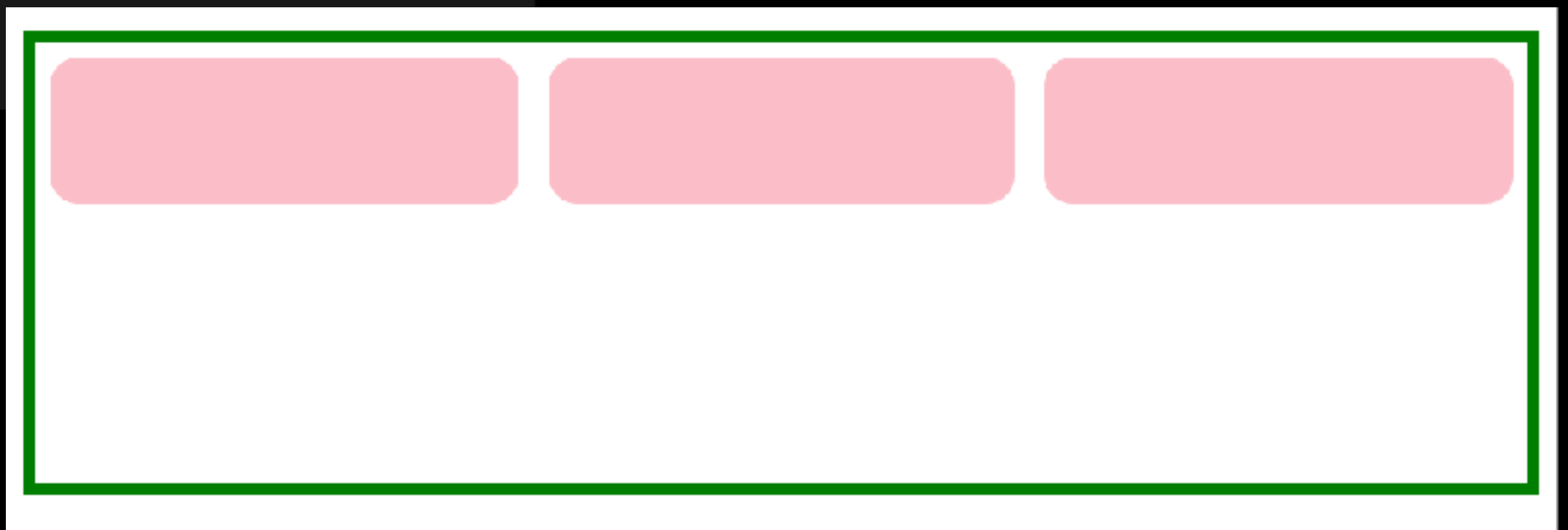
Flex items have **flex-shrink: 1** by default.

*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;  
}
```

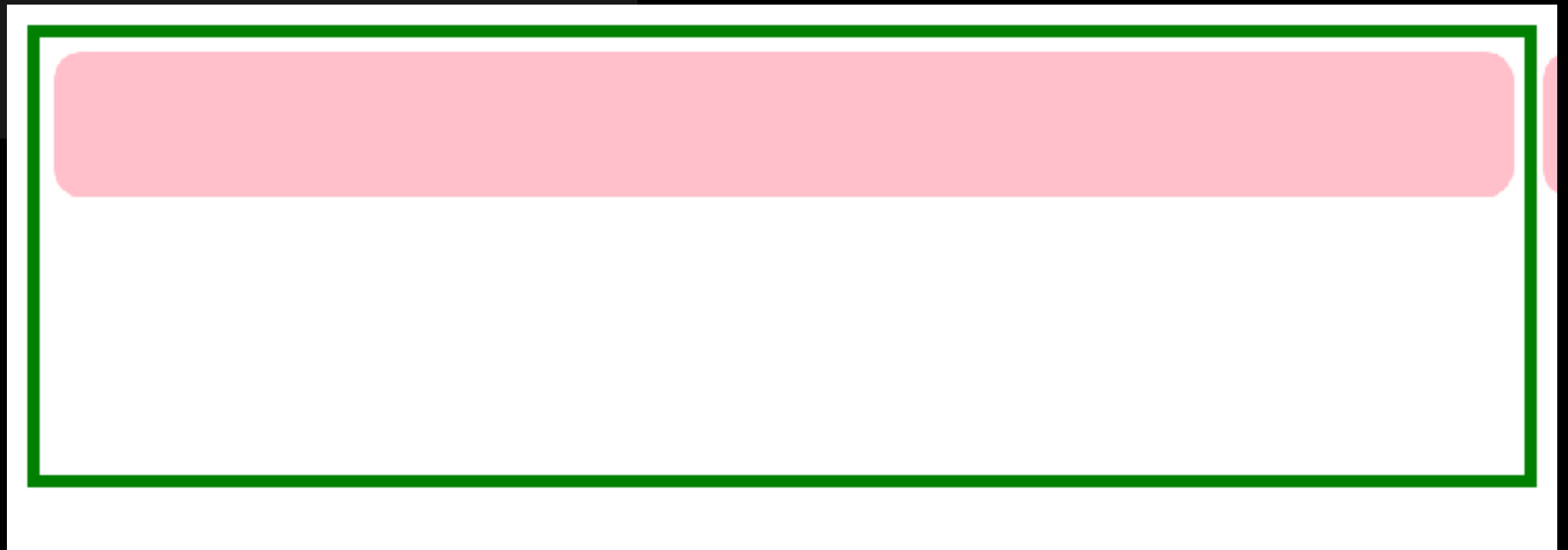
The flex items' widths all shrink to fit the width of the container.



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;
}

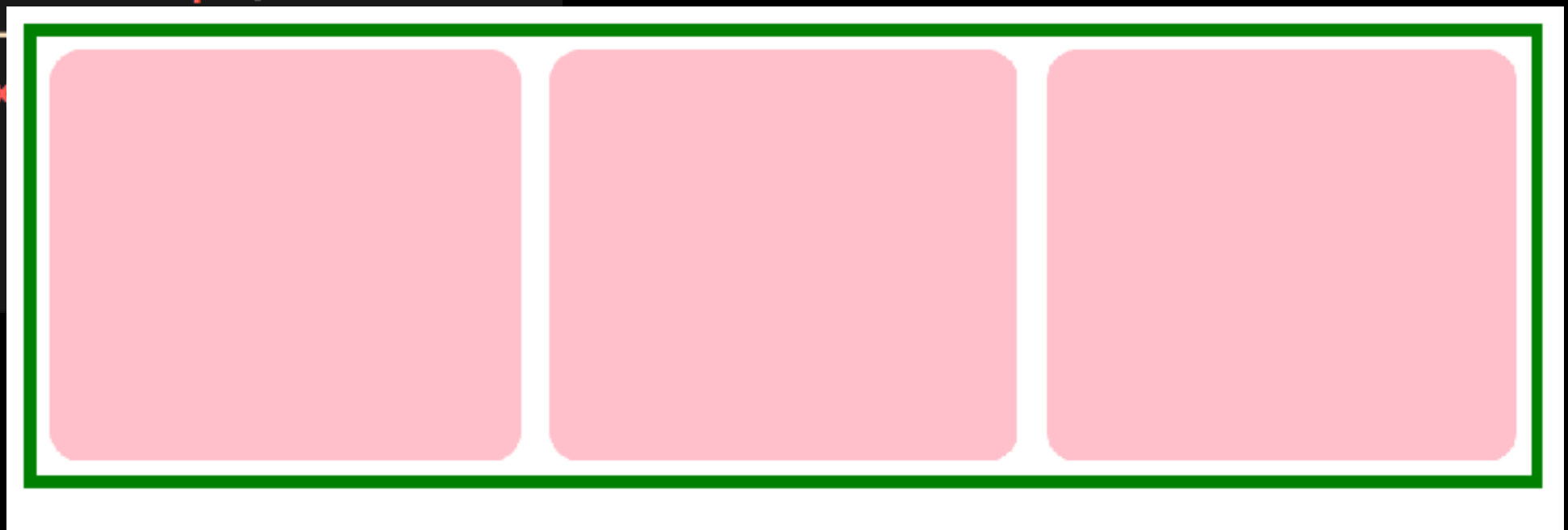
.flexThing {
  border-radius: 10px;
  background-color: pink;
  margin: 5px;
}
```



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-color: #FFB6C1;  
  margin: 5px;  
}
```

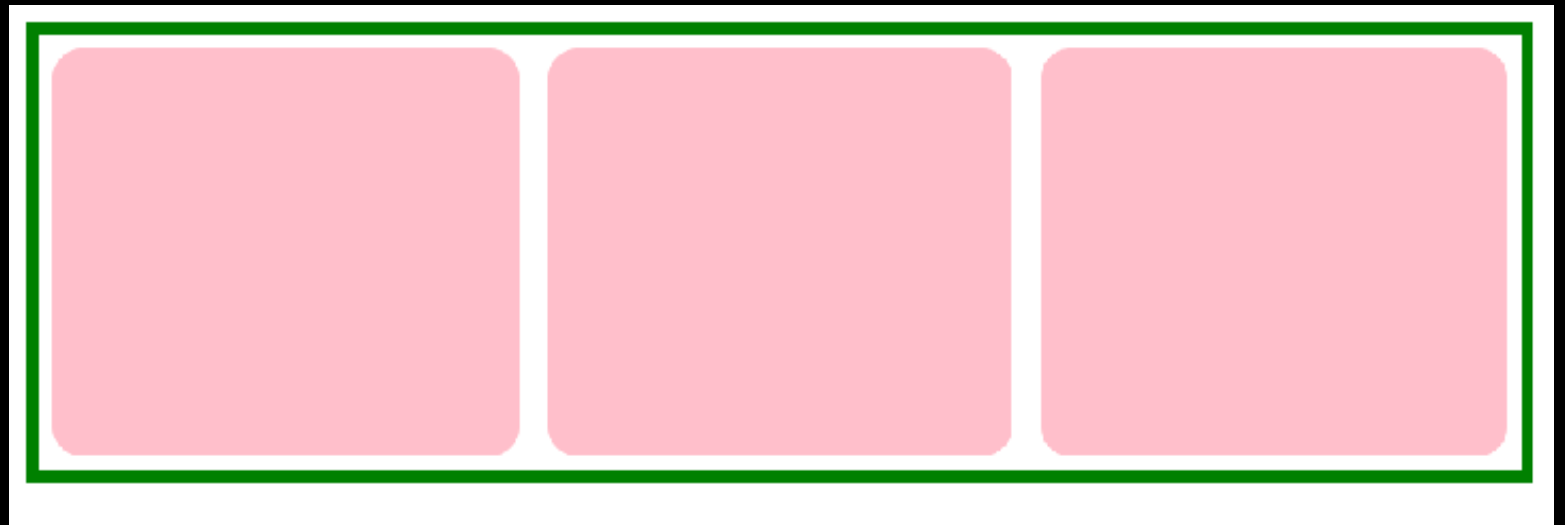


align-items: stretch;

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;  
}
```



*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:

```
▼ #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.

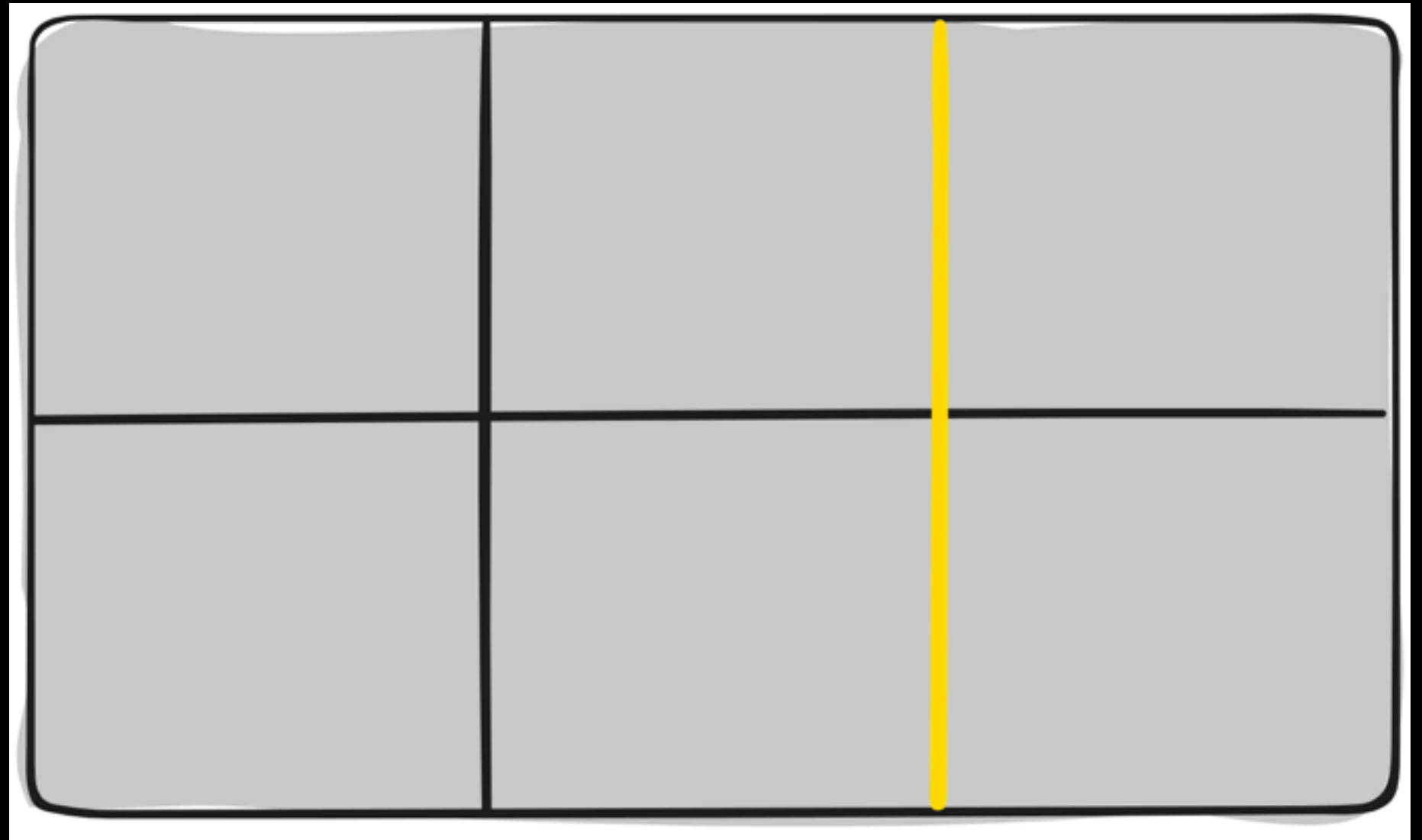
Grid track

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.

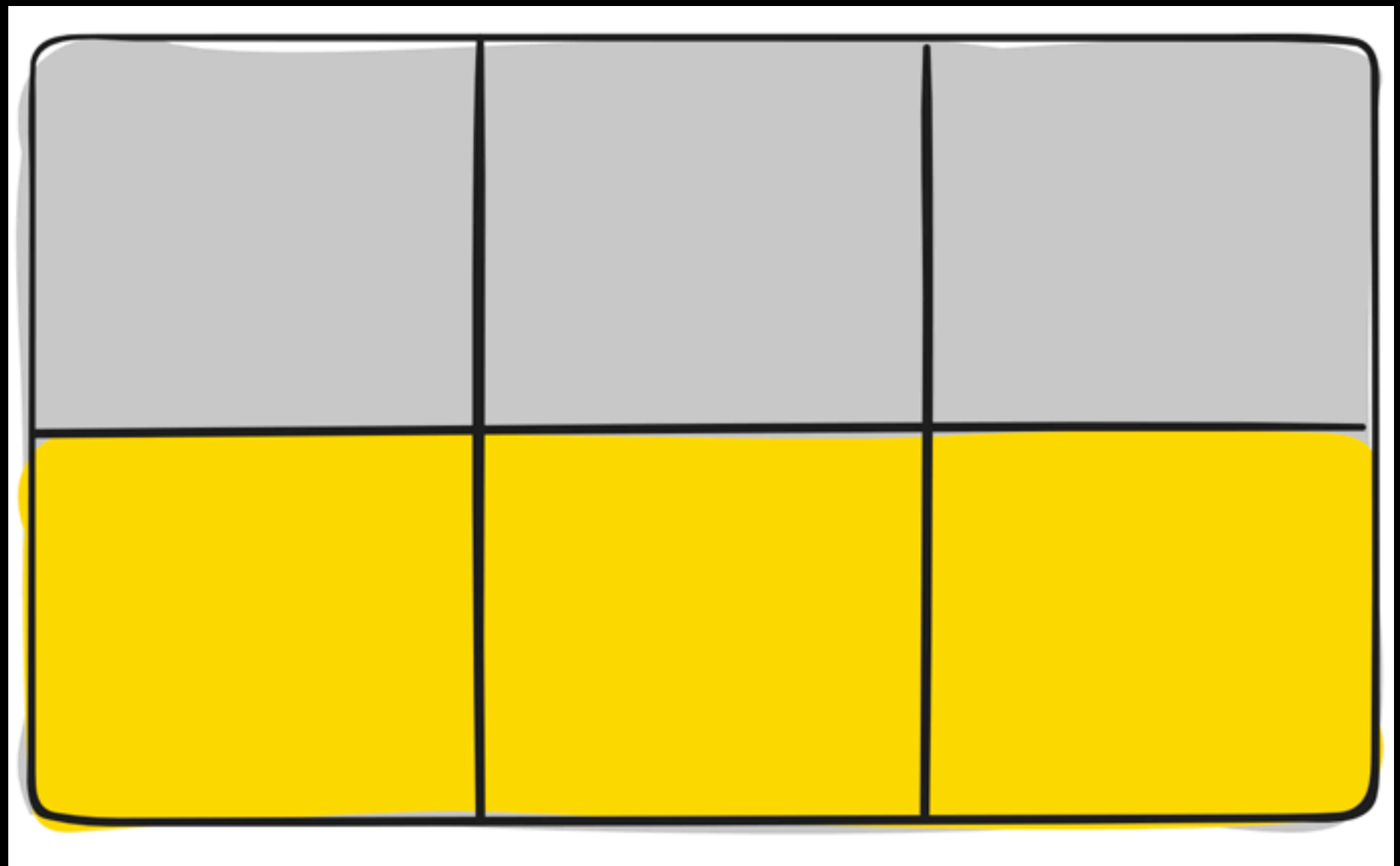
Grid line

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.



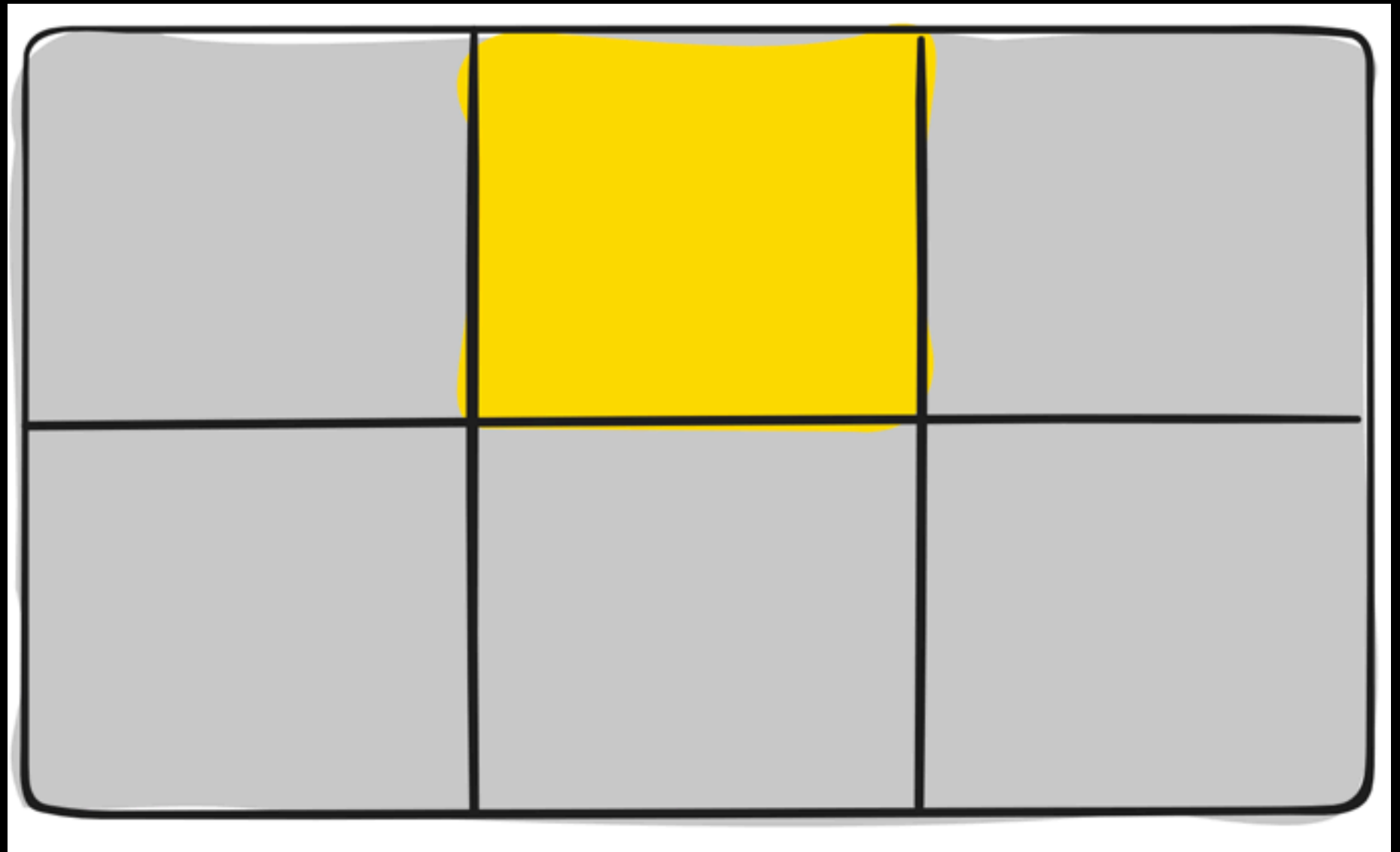
Grid track

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 Properties

grid-column-start

grid-column-end

grid-row-start

grid-row-end

grid-column

grid-row

grid-area

justify-self

align-self

place-self