LEARN MODERN CSS

BY FLAVIO COPES

HTTPS://FLAVIOCOPES.COM

UPDATED MAY 19, 2018

Modern CSS Features

- CSS Grid
- Flexbox
- CSS Custom Properties
- PostCSS

Animate with CSS

- CSS Transitions
- CSS Animations

Modern CSS tips and tricks

- How to center things in modern CSS
- The CSS margin property
- CSS System Fonts
- Style CSS for print

FLEXBOX

Flexbox, also called Flexible Box Module, is one of the two modern layouts systems, along with CSS Grid

- Introduction
- Browser support
- Enable Flexbox
- Container properties

Introduction

Flexbox, also called Flexible Box Module, is one of the two modern layouts systems, along with CSS Grid.

Compared to CSS Grid (which is bi-dimensional), flexbox is a **one-dimensional layout model**. It will control the layout based on a row or on a column, but not together at the same time.

The main goal of flexbox is to allow items to fill the whole space offered by their container, depending on some rules you set.

Unless you need to support old browsers like IE8 and IE9, Flexbox is the tool that lets you forget about using

- Table layouts
- Floats
- clearfix hacks
- `display: table` hacks

Let's dive into flexbox and become a master of it in a very short time.

Browser support

At the time of writing (Feb 2018), it's supported by 97.66% of the users, all the most important browsers implement it since years, so even older browsers (including IE10+) are covered:

IE	Edge*	Firefox	Chrome	Safari	Opera	iOS Safari*	Opera Mini*	Android*
		53	59	7.1	44	8		3
6		54	60	8	45	8.4		4
7	12	55	61	9	46	9.2		4.1
8	13	56	62	9.1	47	9.3		4.3
9	14	57	63	10	48	10.2		4.4
10	15	58	64	10.1	49	10.3		4.4.4
11	16	59	65	11	50	11.2	all	62
	17	60	66	11.1	51	11.3		
	18	61	67	TP	52			
			68					

While we must wait a few years for users to catch up on CSS Grid, Flexbox is an older technology and can be used right now.

Enable Flexbox

A flexbox layout is applied to a container, by setting

```
display: flex;
```

or

```
display: inline-flex;
```

the content **inside the container** will be aligned using flexbox.

Container properties

Some flexbox properties apply to the container, which sets the general rules for its items. They are

- `flex-direction`
- `justify-content`
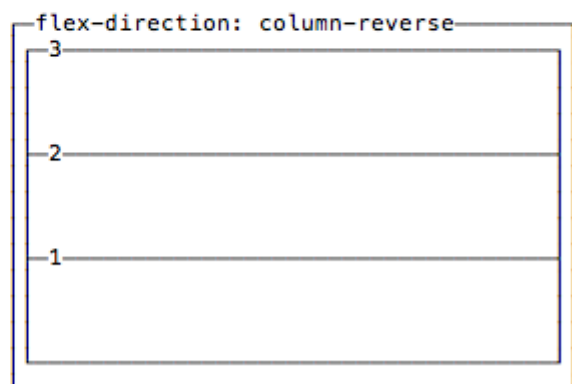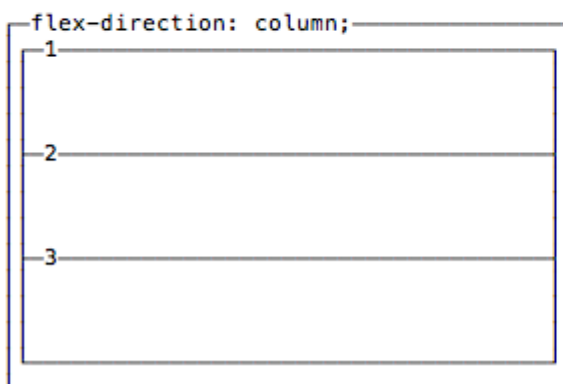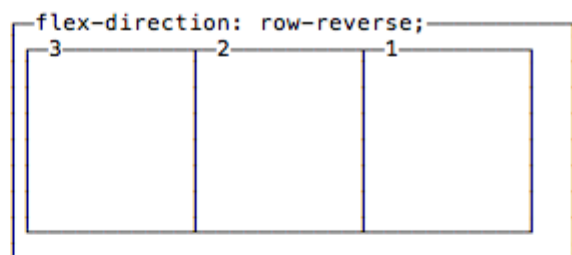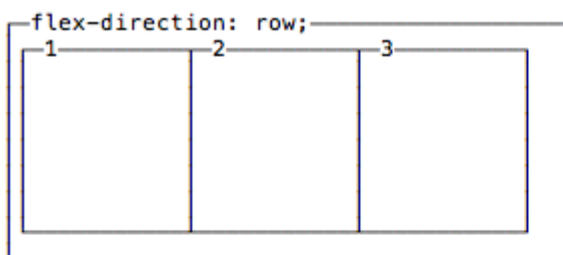- `align-items`
- `flex-wrap`
- `flex-flow`

Align rows or columns

The first property we see, **`flex-direction`**, determines if the container should align its items as rows, or as columns:

- `flex-direction: row` places items as a **row**, in the text direction (left-to-right for western countries)
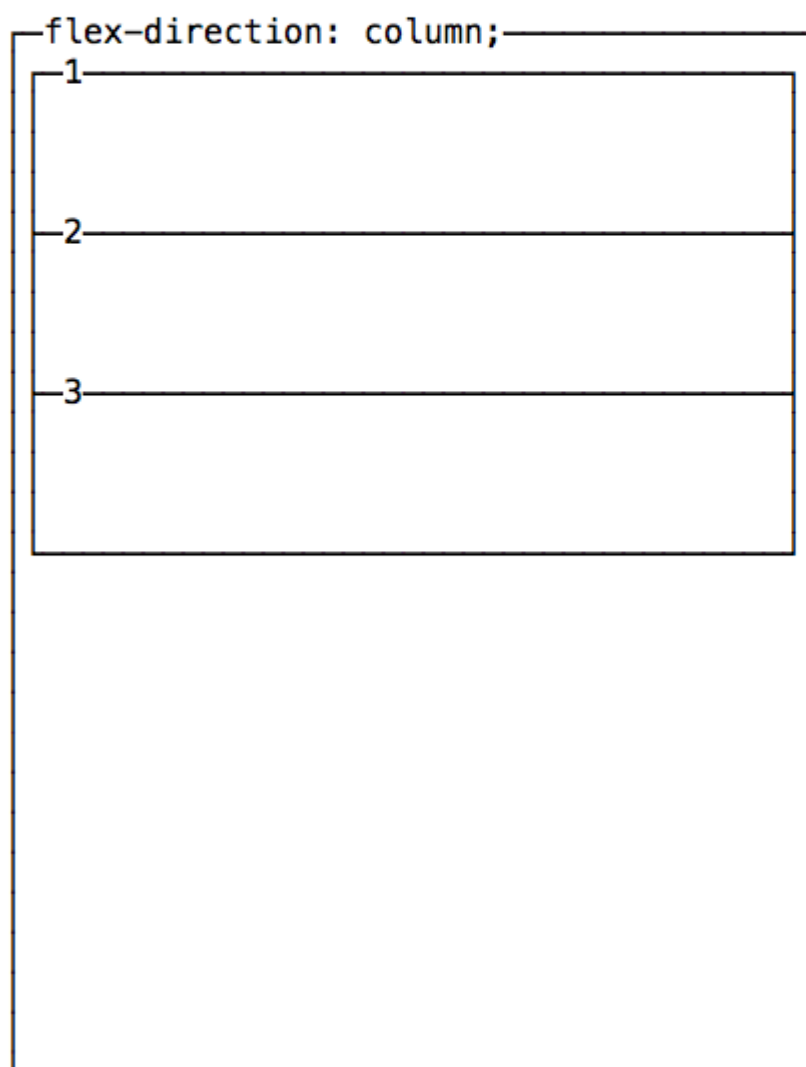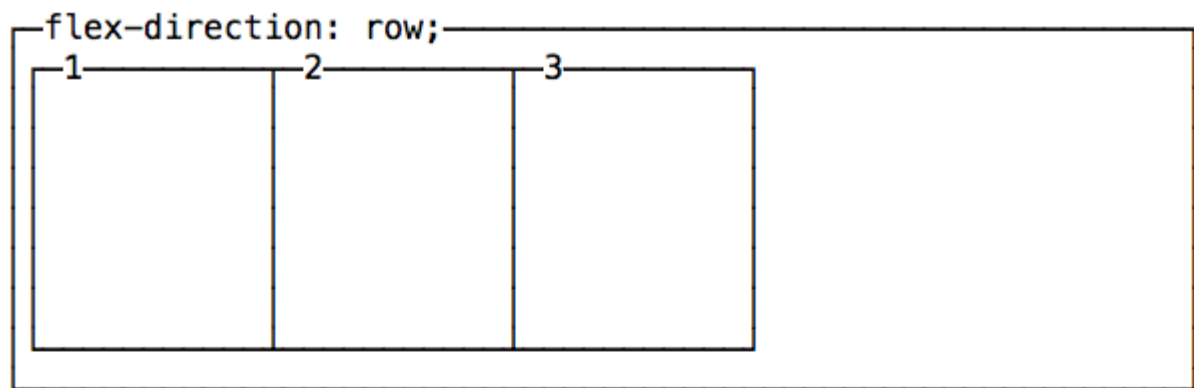
- `flex-direction: row-reverse` places items just like `row` but in the opposite direction
- `flex-direction: column` places items in a **column**, ordering top to bottom
- `flex-direction: column-reverse` places items in a column, just like `column` but in the opposite direction

```
┌flex-direction: row;──────────┐
│ ┌1────────┬2────────┬3──────┐ │
│ │         │         │       │ │
│ │         │         │       │ │
│ │         │         │       │ │
│ │         │         │       │ │
│ └─────────┴─────────┴───────┘ │
└───────────────────────────────┘
```

```
┌flex-direction: row-reverse;──┐
│ ┌3────────┬2────────┬1──────┐ │
│ │         │         │       │ │
│ │         │         │       │ │
│ │         │         │       │ │
│ │         │         │       │ │
│ └─────────┴─────────┴───────┘ │
└───────────────────────────────┘
```

```
┌flex-direction: column;───────┐
│ ┌1────────────────────────┐ │
│ ├2────────────────────────┤ │
│ │                         │ │
│ ├3────────────────────────┤ │
│ │                         │ │
│ └─────────────────────────┘ │
└───────────────────────────────┘
```

```
┌flex-direction: column-reverse┐
│ ┌3────────────────────────┐ │
│ ├2────────────────────────┤ │
│ │                         │ │
│ ├1────────────────────────┤ │
│ │                         │ │
│ └─────────────────────────┘ │
└───────────────────────────────┘
```

Vertical and horizontal alignment

By default items start from the left is `flex-direction` is row, and from the top if `flex-direction` is column.

```
┌─flex-direction: row;─────────────────────────────────────────┐
│ ┌─1──────────────┬─2──────────────┬─3──────────────┐         │
│ │                │                │                │         │
│ │                │                │                │         │
│ │                │                │                │         │
│ │                │                │                │         │
│ │                │                │                │         │
│ └────────────────┴────────────────┴────────────────┘         │
└───────────────────────────────────────────────────────────────┘

┌─flex-direction: column;──────────────────────┐
│ ┌─1──────────────────────────────────────────┐ │
│ │                                            │ │
│ ├─2──────────────────────────────────────────┤ │
│ │                                            │ │
│ ├─3──────────────────────────────────────────┤ │
│ │                                            │ │
│ └────────────────────────────────────────────┘ │
│                                                │
│                                                │
│                                                │
│                                                │
│                                                │
│                                                │
│                                                │
└────────────────────────────────────────────────┘
```
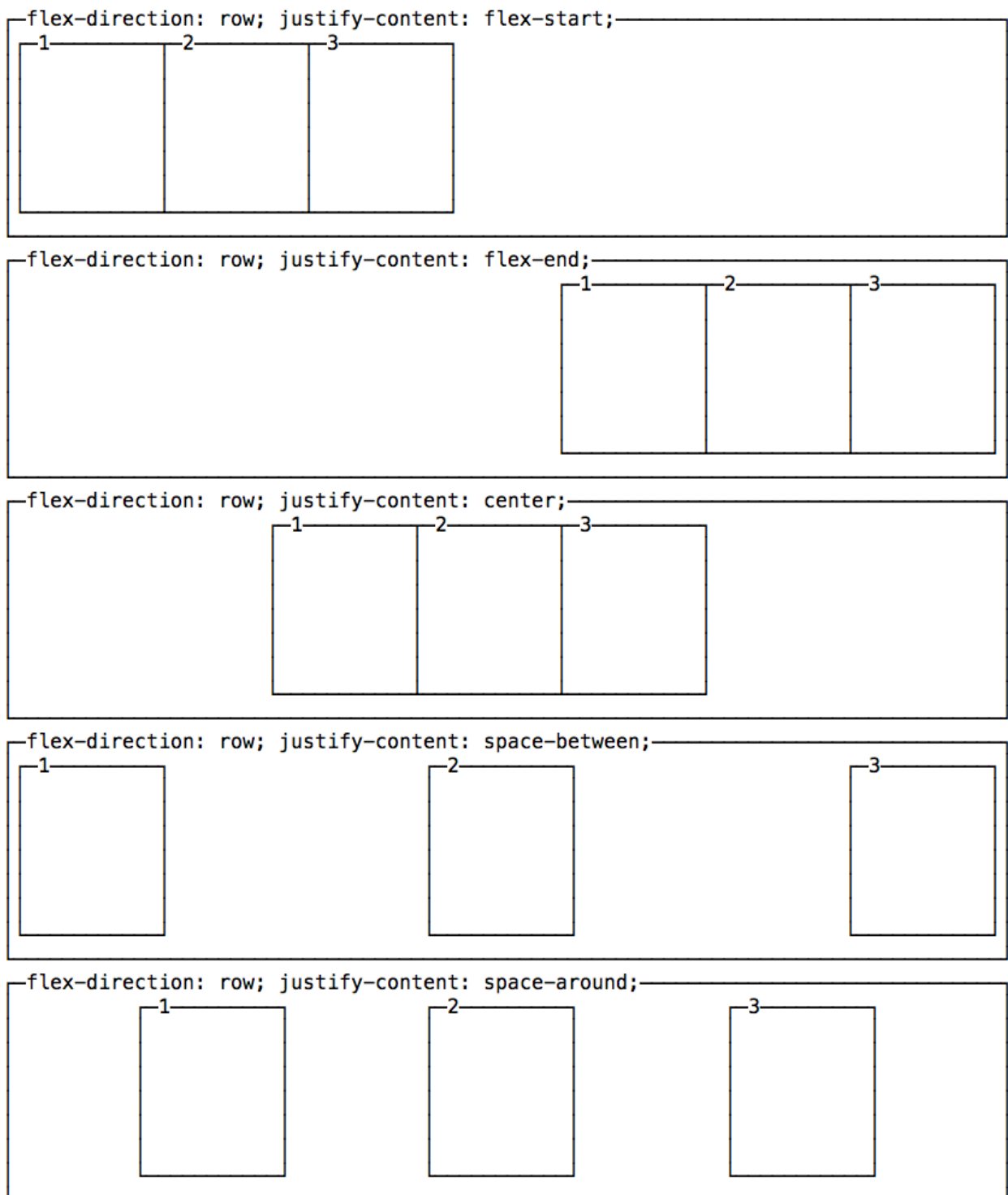
You can change this behavior using `justify-content` to change the horizontal alignment, and `align-items` to change the vertical alignment.

Change the horizontal alignment

`justify-content` has 5 possible values:

- `flex-start` : align to the left side of the container.
- `flex-end` : align to the right side of the container.
- `center` : align at the center of the container.
- `space-between` : display with equal spacing between them.
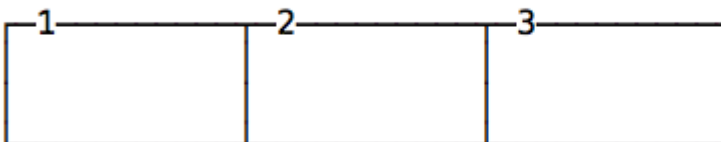- `space-around` : display with equal spacing around them

```
┌─flex-direction: row; justify-content: flex-start;─────────────────────────
│ ┌1──────────┬2──────────┬3──────────┐
│ │           │           │           │
│ │           │           │           │
│ │           │           │           │
│ │           │           │           │
│ └───────────┴───────────┴───────────┘
└────────────────────────────────────────────────────────────────────────────

┌─flex-direction: row; justify-content: flex-end;──────────────────────────
│                              ┌1──────────┬2──────────┬3──────────┐
│                              │           │           │           │
│                              │           │           │           │
│                              │           │           │           │
│                              │           │           │           │
│                              └───────────┴───────────┴───────────┘
└────────────────────────────────────────────────────────────────────────────

┌─flex-direction: row; justify-content: center;────────────────────────────
│              ┌1──────────┬2──────────┬3──────────┐
│              │           │           │           │
│              │           │           │           │
│              │           │           │           │
│              │           │           │           │
│              └───────────┴───────────┴───────────┘
└────────────────────────────────────────────────────────────────────────────

┌─flex-direction: row; justify-content: space-between;─────────────────────
│ ┌1──────────┐          ┌2──────────┐          ┌3──────────┐
│ │           │          │           │          │           │
│ │           │          │           │          │           │
│ │           │          │           │          │           │
│ └───────────┘          └───────────┘          └───────────┘
└────────────────────────────────────────────────────────────────────────────

┌─flex-direction: row; justify-content: space-around;──────────────────────
│     ┌1──────────┐          ┌2──────────┐          ┌3──────────┐
│     │           │          │           │          │           │
│     │           │          │           │          │           │
│     │           │          │           │          │           │
│     └───────────┘          └───────────┘          └───────────┘
└────────────────────────────────────────────────────────────────────────────
```

Change the vertical alignment

`align-items` has 5 possible values:

- `flex-start` : align to the top of the container.
- `flex-end` : align to the bottom of the container.

- `center` : align at the vertical center of the container.

- `baseline` : display at the baseline of the container.

- `stretch` : items are stretched to fit the container.
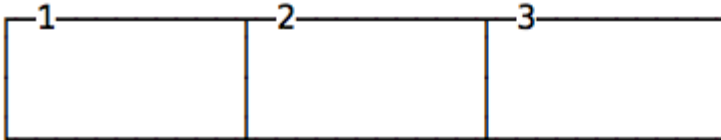
```
┌─flex-direction: row;── align-items: flex-start;────────────┐
│ ┌─1──────────┬─2──────────┬─3──────────┐                    │
│ │            │            │            │                    │
│ │            │            │            │                    │
│ └────────────┴────────────┴────────────┘                    │
│                                                             │
│                                                             │
│                                                             │
│                                                             │
│                                                             │
└─────────────────────────────────────────────────────────────┘
┌─flex-direction: row;── align-items: flex-end;──────────────┐
│                                                             │
│                                                             │
│                                                             │
│                                                             │
│                                                             │
│ ┌─1──────────┬─2──────────┬─3──────────┐                    │
│ │            │            │            │                    │
│ │            │            │            │                    │
│ └────────────┴────────────┴────────────┘                    │
└─────────────────────────────────────────────────────────────┘
┌─flex-direction: row;── align-items: baseline;──────────────┐
│ ┌─1──────────┬─2──────────┬─3──────────┐                    │
│ │            │            │            │                    │
│ │            │            │            │                    │
│ └────────────┴────────────┴────────────┘                    │
│                                                             │
│                                                             │
│                                                             │
│                                                             │
└─────────────────────────────────────────────────────────────┘
┌─flex-direction: row;── align-items: stretch;───────────────┐
│ ┌─1──────────┬─2──────────┬─3──────────┐                    │
│ │            │            │            │                    │
│ │            │            │            │                    │
```
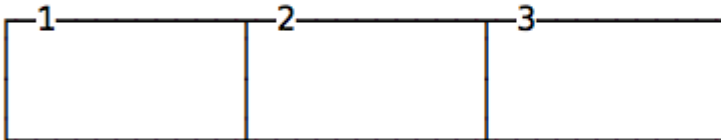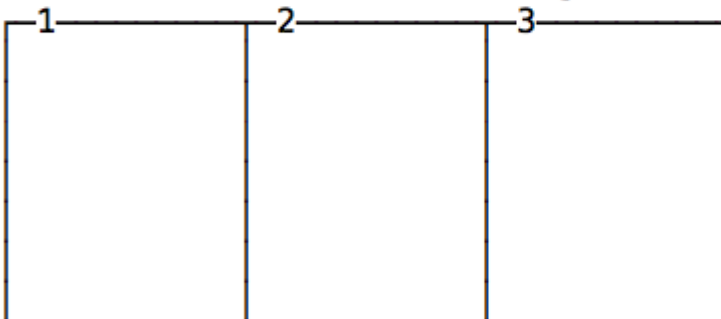
```
 ┌────────────┬──────────┬───────────┐                          │
 │            │          │           │                          │
 ├──flex-direction: row;── align-items: center;────────────────┐
 │                                                             │
 │                                                             │
 │                                                             │
 │   ┌─1──────────┬─2──────────┬─3──────────┐                  │
 │   │            │            │            │                  │
 │   │            │            │            │                  │
 │   └────────────┴────────────┴────────────┘                  │
 │                                                             │
 │                                                             │
 │                                                             │
 └─────────────────────────────────────────────────────────────┘
```
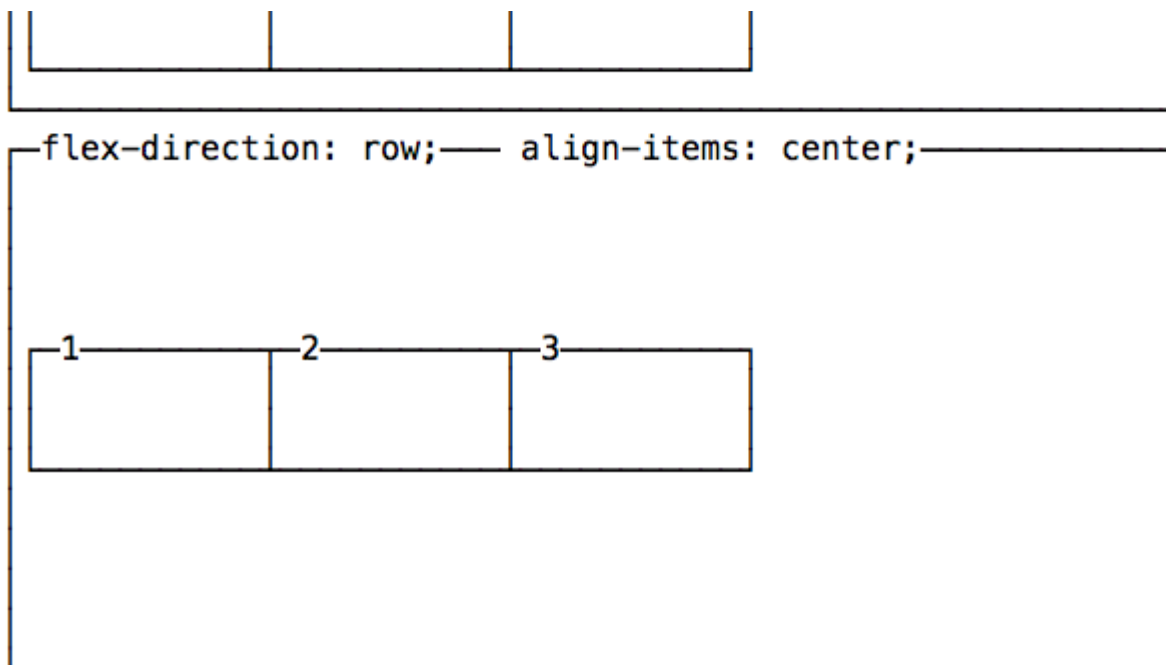
A note on `baseline`

`baseline` looks similar to `flex-start` in this example, due to my boxes being too simple. Check out this Codepen (https://codepen.io/flaviocopes/pen/oExoJR) to have a more useful example, which I forked from a Pen originally created by Martin Michálek (https://twitter.com/machal) . As you can see there, items dimensions are aligned.

Wrap

By default items in a flexbox container are kept on a single line, shrinking them to fit in the container.

To force the items to spread across multiple lines, use `flex-wrap: wrap` . This will distribute the items according to the order set in

`flex-direction`. Use `flex-wrap: wrap-reverse` to reverse this order.

A shorthand property called `flex-flow` allows you to specify `flex-direction` and `flex-wrap` in a single line, by adding the `flex-direction` value first, followed by `flex-wrap` value, for example: `flex-flow: row wrap`.

Properties that apply to each single item

Since now, we've seen the properties you can apply to the container.

Single items can have a certain amount of independence and flexibility, and you can alter their appearance using those properties:

- `order`
- `align-self`
- `flex-grow`
- `flex-shrink`
- `flex-basis`
- `flex`

Let's see them in details.

Moving items before / after another one using order

Items are ordered based on a order they are assigned. By default every item has order `0` and the appearance in the HTML determines the final order.

You can override this property using `order` on each separate item. This is a property you set on the item, not the container. You can make an item appear before all the others by setting a negative value.
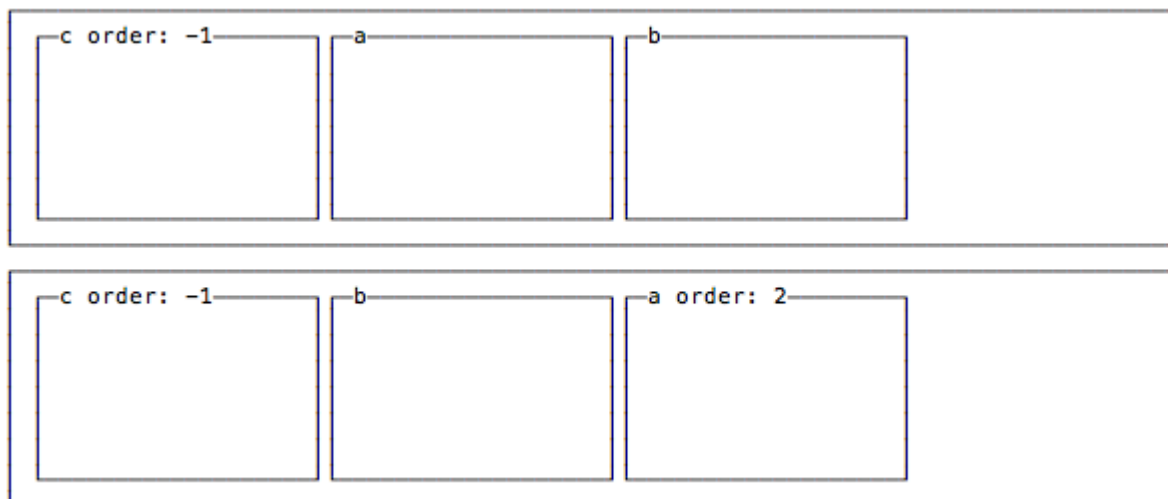
Vertical alignment using align-self

An item can choose to **override** the container `align-items` setting, using **align-self**, which has the same 5 possible values of `align-items`:

- `flex-start` : align to the top of the container.

- `flex-end` : align to the bottom of the container.

- `center` : align at the vertical center of the container.

- `baseline` : display at the baseline of the container.

- `stretch` : items are stretched to fit the container.

```
┌flex-direction: row; align-items: flex-start;──────────┐
│┌1─────────┐                          ┌3─────────┐     │
││          │                          │          │     │
││          │                          │          │     │
│└──────────┘                          └──────────┘     │
│                                                        │
│                                                        │
│                                                        │
│              ┌align-self: flex-end─┐                   │
│              │                     │                   │
│              │                     │                   │
│              └─────────────────────┘                   │
│                                                        │
└────────────────────────────────────────────────────────┘
```
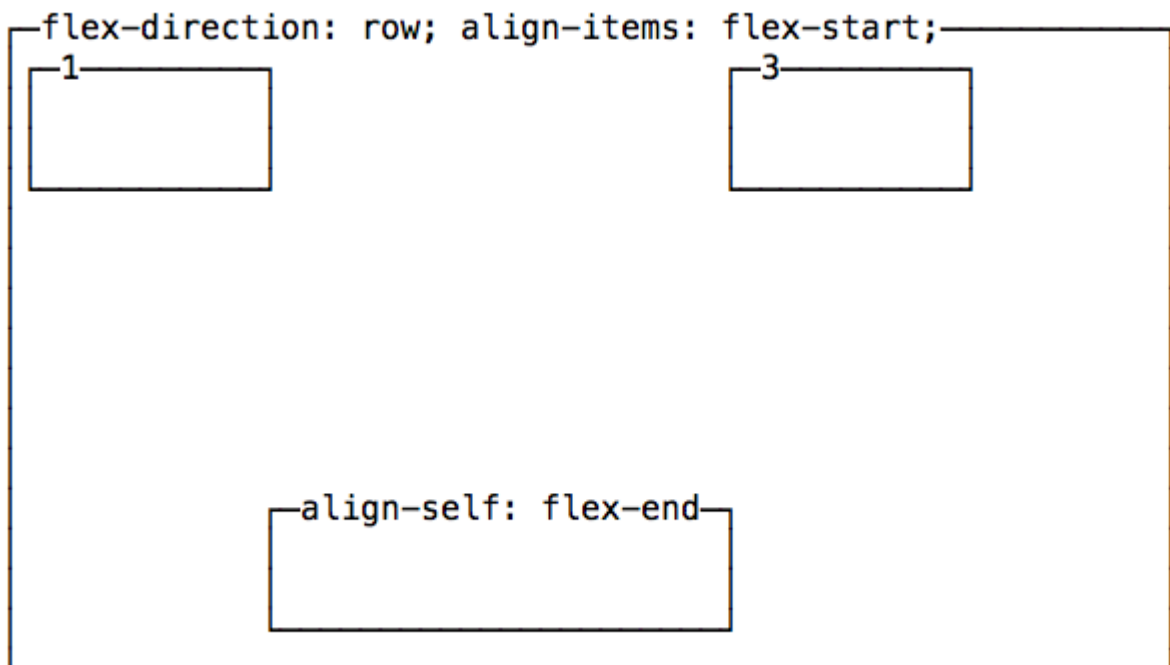
Grow or shrink an item if necessary

flex-grow

The defaut for any item is 0.

If all items are defined as 1 and one is defined as 2, the bigger element will take the space of two "1" items.

flex-shrink

The defaut for any item is 1.

If all items are defined as 1 and one is defined as 3, the bigger element will shrink 3x the other ones. When less space is available, it will take 3x less space.

flex-basis

If set to `auto`, it sizes an item according to its width or height, and adds extra space based on the `flex-grow` property.

If set to 0, it does not add any extra space for the item when calculating the layout.

If you specify a pixel number value, it will use that as the length value (width or height depends if it's a row or a column item)

flex

This property combines the above 3 properties:

- `flex-grow`
- `flex-shrink`
- `flex-basis`

and provides a shorthand syntax: `flex: 0 1 auto`

CSS GRID

CSS Grid is the new kid in the CSS town, and while not yet fully supported by all browsers, it's going to be the future system for layouts
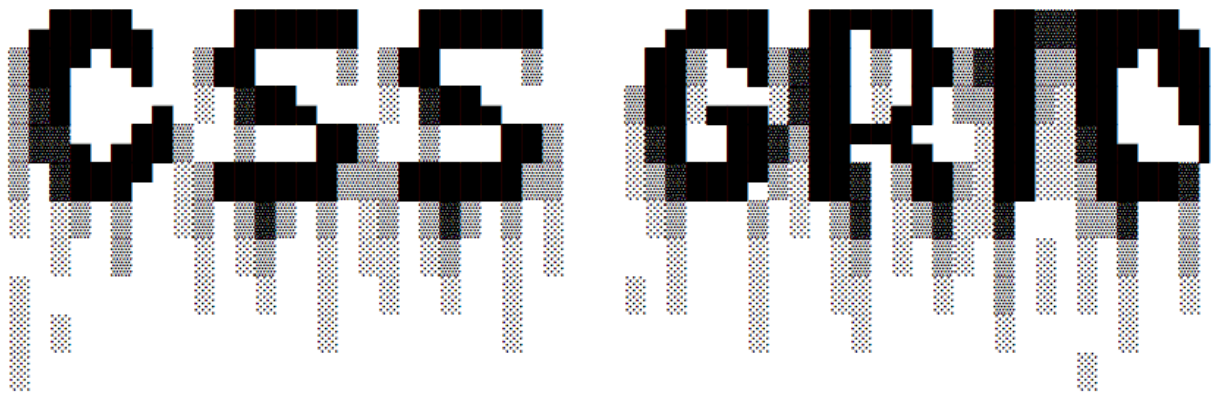
> *The Grid. A digital frontier. I tried to picture clusters of information as they moved through the computer. What did they look like? Ships? Motorcycles? Were the circuits like freeways? I*

kept dreaming of a world I thought I'd never see. And then one day.. I got in. — Tron: Legacy

- Introduction to CSS Grid
- The basics

  - grid-template-columns and grid-template-rows
  - Automatic dimensions
  - Different columns and rows dimensions
  - Adding space between the cells
  - Spawning items on multiple columns and/or rows
  - Shorthand syntax

- More grid configuration

  - Using fractions
  - Using percentages and rem
  - Using `repeat()`
  - Specify a minimum width for a row
  - Positioning elements using `grid-template-areas`
  - Adding empty cells in template areas

- Fill a page with a grid
- Wrapping up

Introduction to CSS Grid

CSS Grid is a fundamentally new approach to building layouts using CSS.

Keep an eye on the CSS Grid Layout page on caniuse.com (https://caniuse.com/#feat=css-grid) to find out which browsers currently support it. At the time of writing, Feb 2018, all major browsers (except IE, which will never have support for it) are already supporting this technology, covering 78% of all users.

CSS Grid is not a competitor to Flexbox. They interoperate and collaborate on complex layouts, because CSS Grid works on 2 dimensions (rows AND columns) while Flexbox works on a single dimension (rows OR columns).

Building layouts for the web has traditionally been a complicated topic.

I won't dig into the reasons for this complexity, which is a complex topic on its own, but you can think yourself as a very lucky human because nowadays **you have 2 very powerful and well supported tools at your disposal**:

- **CSS Flexbox**
- **CSS Grid**

These 2 are the tools to build the Web layouts of the future.

Unless you need to support old browsers like IE8 and IE9, there is no reason to be messing with things like:

- Table layouts
- Floats
- clearfix hacks
- `display: table` hacks

In this guide there's all you need to know about going from a zero knowledge of CSS Grid to being a proficient user.

The basics

The CSS Grid layout is activated on a container element (which can be a `div` or any other tag) by setting `display: grid`.

As with flexbox, you can define some properties on the container, and some properties on each individual item in the grid.

These properties combined will determine the final look of the grid.

The most basic container properties are `grid-template-columns` and `grid-template-rows`.

grid-template-columns and grid-template-rows

Those properties define the number of columns and rows in the grid, and they also set the width of each column/row.

The following snippet defines a grid with 4 columns each 200px wide, and 2 rows with a 300px height each.

```css
.container {
  display: grid;
  grid-template-columns: 200px 200px 200px 200px;
  grid-template-rows: 300px 300px;
}
```
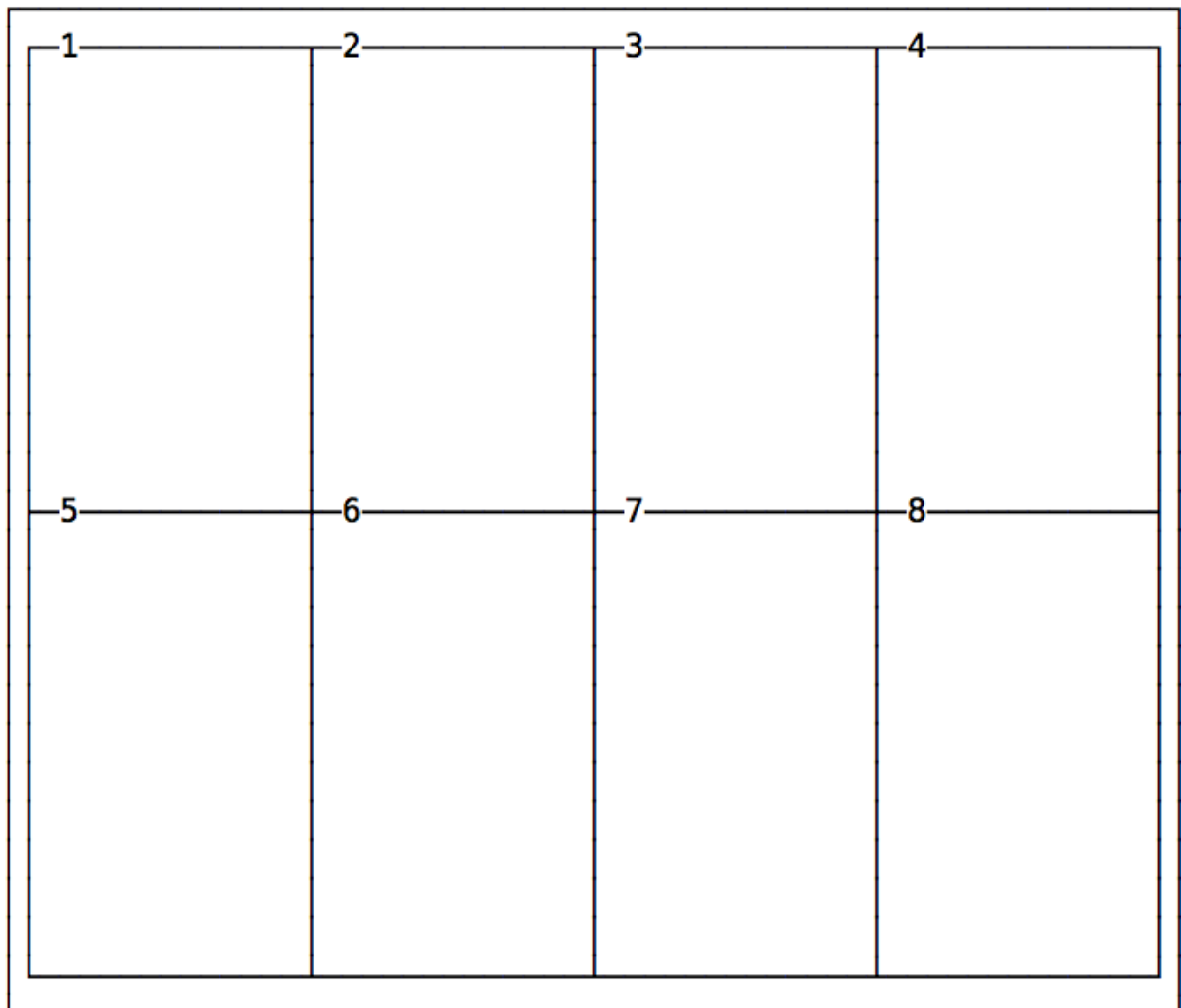
Here's another example of a grid with 2 columns and 2 rows:

```
.container {
  display: grid;
  grid-template-columns: 200px 200px;
  grid-template-rows: 100px 100px;
}
```
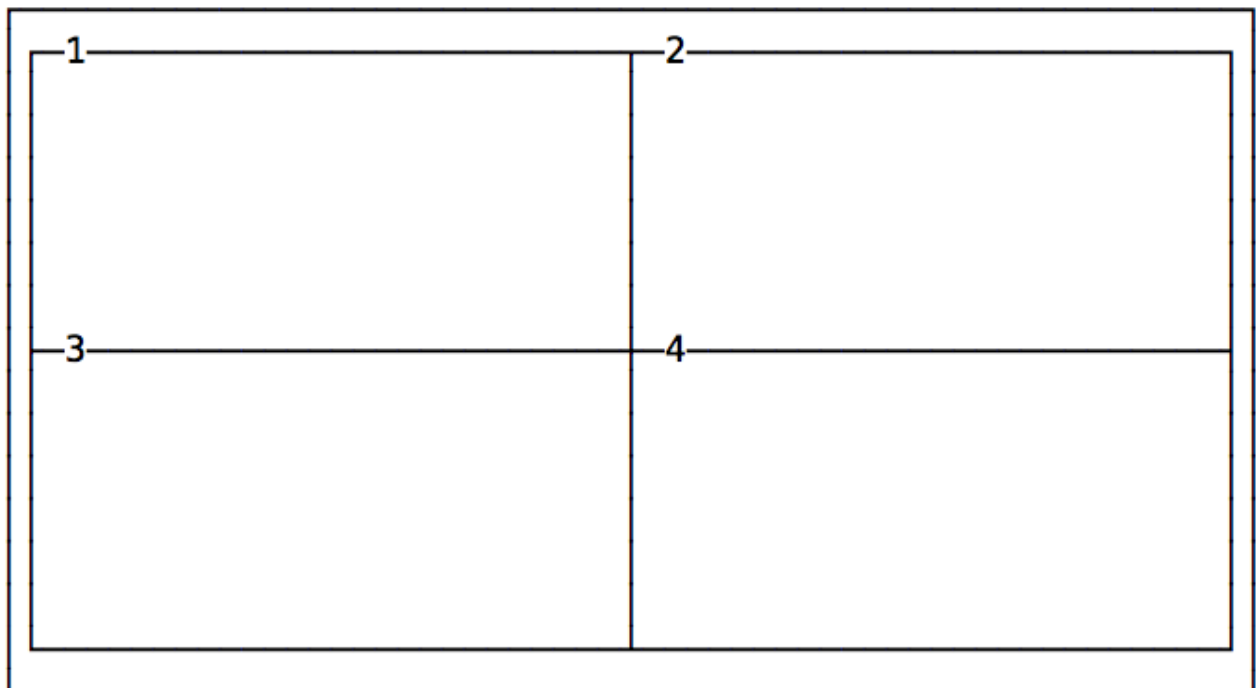

Automatic dimensions

Many times you might have a fixed header size, a fixed footer size, and the main content that is flexible in height, depending on its length. In this case you can use the `auto` keyword:

```
.container {
  display: grid;
```

```
    grid-template-rows: 100px auto 100px;
}
```

Different columns and rows dimensions

In the above examples we made regular grids by using the same values for rows and the same values for columns.

You can specify any value for each row/column, to create a lot of different designs:

```
.container {
  display: grid;
  grid-template-columns: 100px 200px;
  grid-template-rows: 100px 50px;
}
```
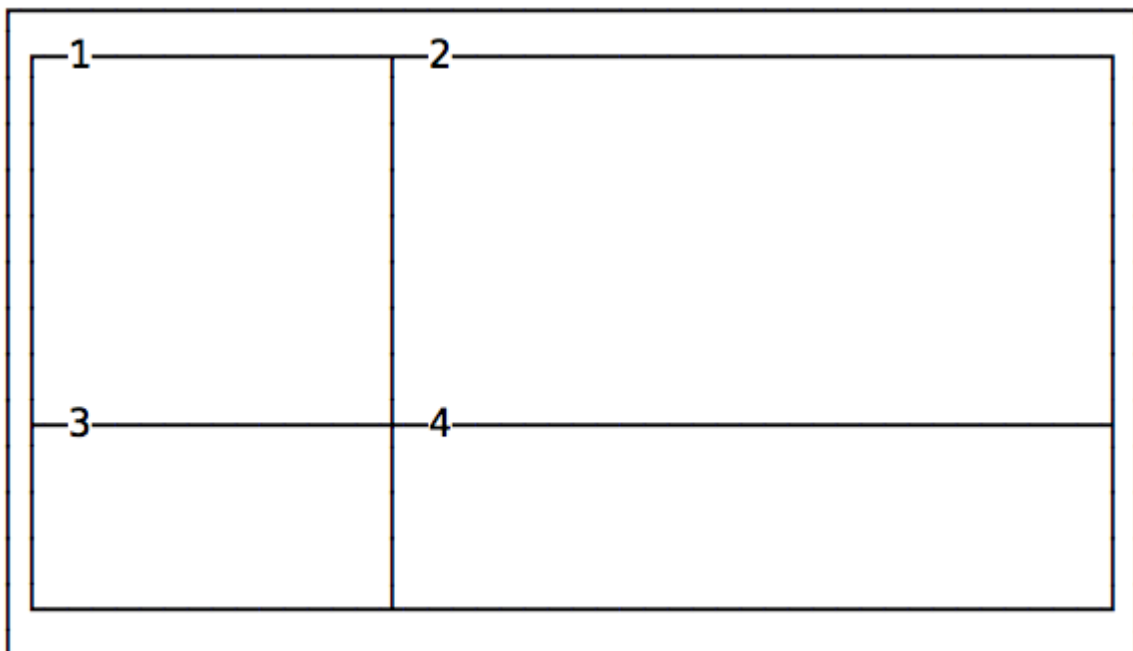
Another example:

```
.container {
  display: grid;
  grid-template-columns: 10px 100px;
  grid-template-rows: 100px 10px;
}
```
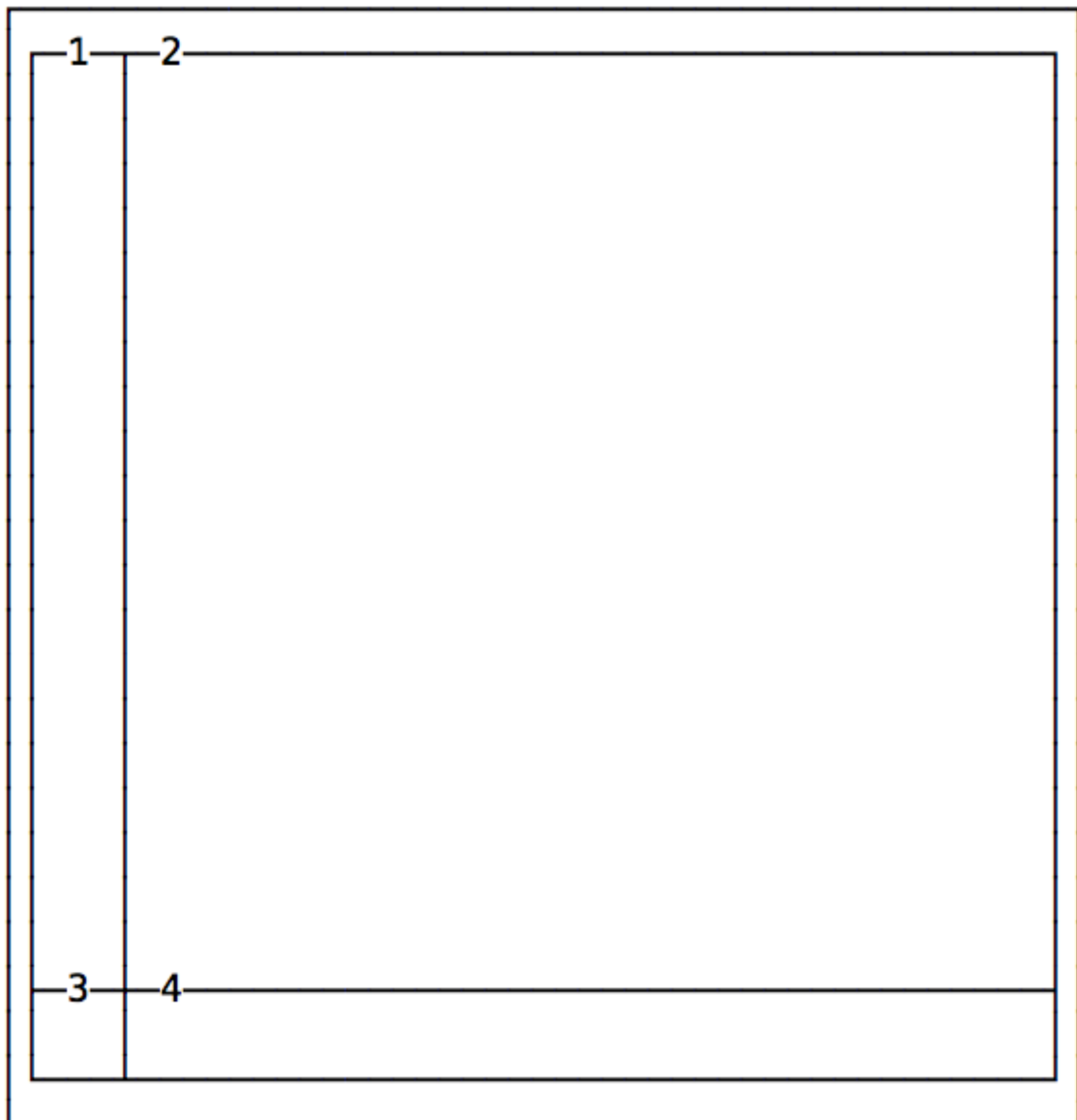
Adding space between the cells

Unless specified, there is no space between the cells.

You can add spacing by using those properties: - `grid-column-gap`
- `grid-row-gap`

or the shorthand syntax `grid-gap`.

Example:

```css
.container {
  display: grid;
  grid-template-columns: 100px 200px;
  grid-template-rows: 100px 50px;
  grid-column-gap: 25px;
  grid-row-gap: 25px;
}
```
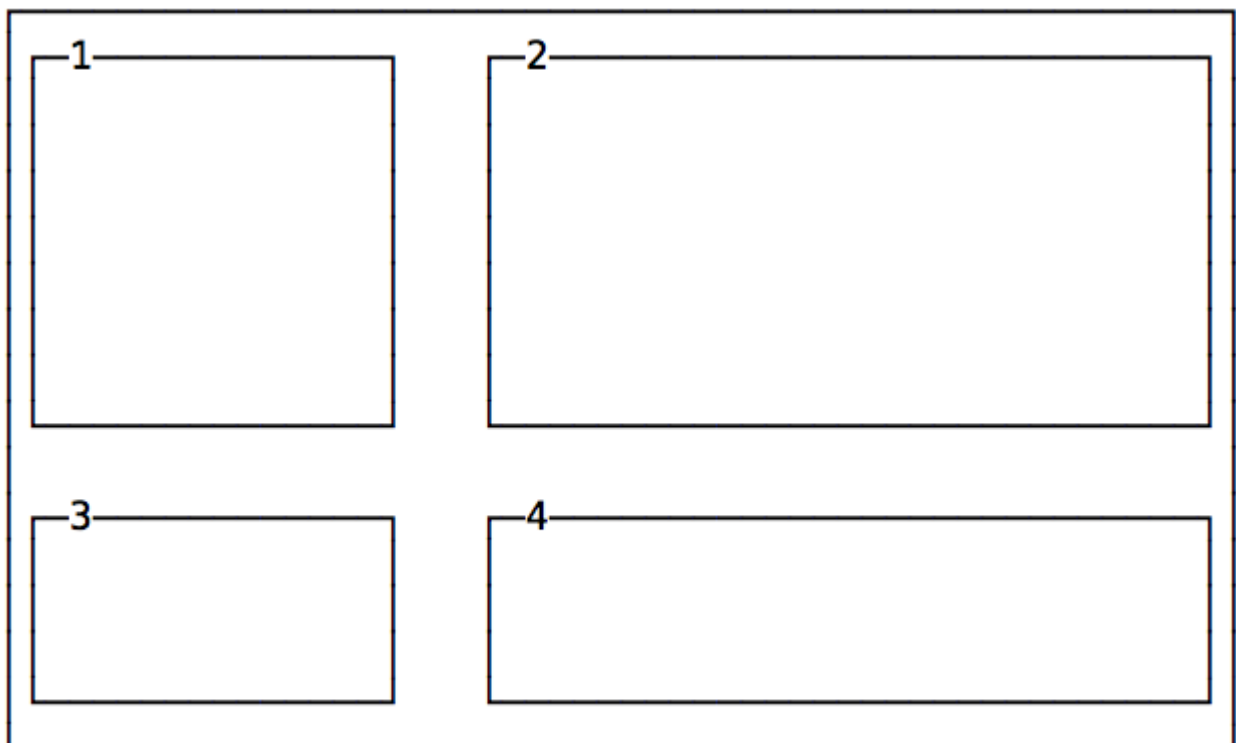
The same layout using the shorthand:

```
.container {
  display: grid;
  grid-template-columns: 100px 200px;
  grid-template-rows: 100px 50px;
  grid-gap: 25px;
}
```

Spawning items on multiple columns and/or rows

Every cell item has the option to occupy more than just one box in the row, and expand horizontally or vertically to get more space, while respecting the grid proportions set in the container.

Those are the properties we'll use for that: - `grid-column-start` - `grid-column-end` - `grid-row-start` - `grid-row-end`

Example:

```
.container {
  display: grid;
  grid-template-columns: 200px 200px 200px 200px;
  grid-template-rows: 300px 300px;
}
.item1 {
  grid-column-start: 2;
  grid-column-end: 4;
}
.item6 {
```

```
    grid-column-start: 3;
    grid-column-end: 5;
}
```
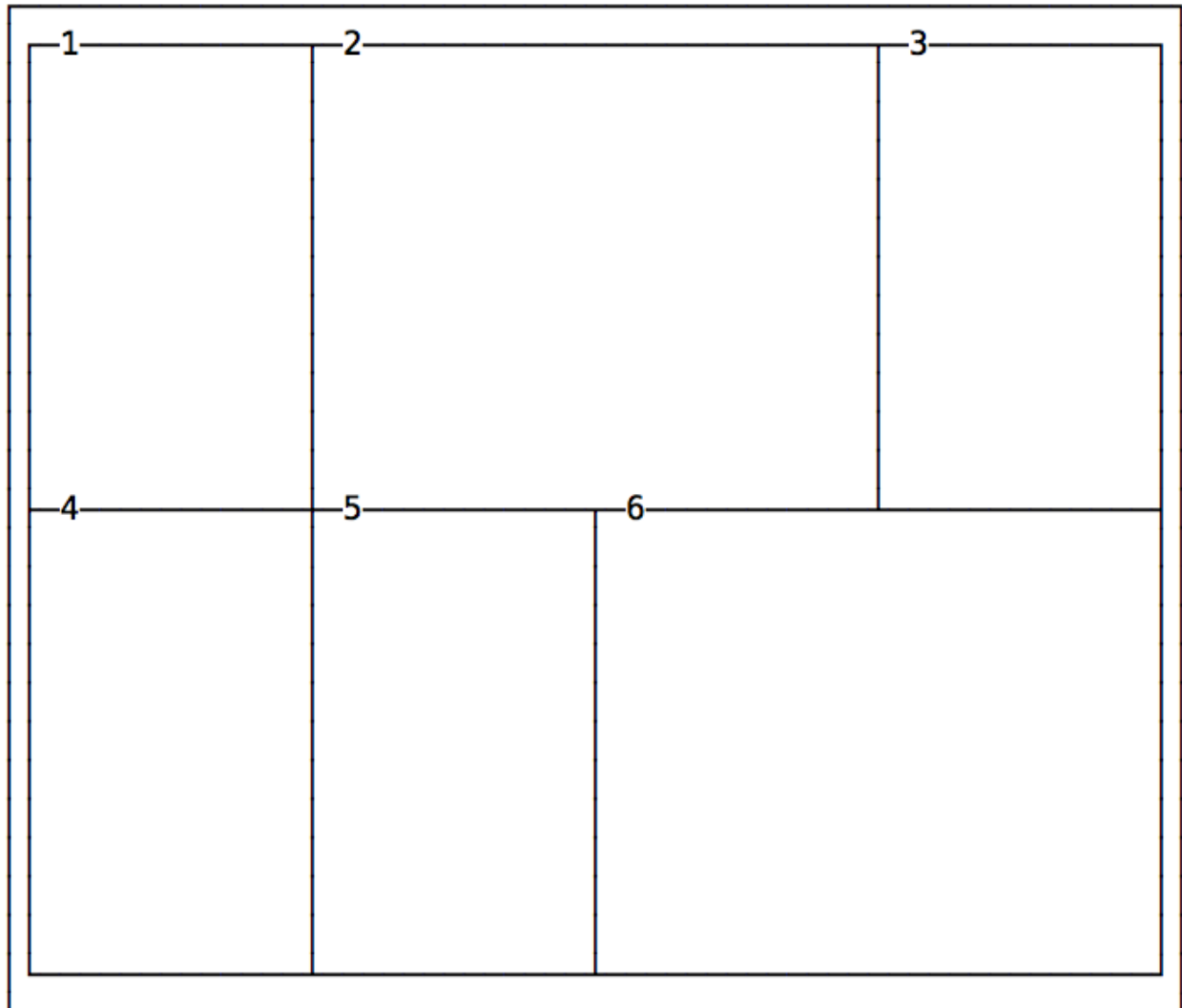

The numbers correspond to the vertical line that separates each column, starting from 1:
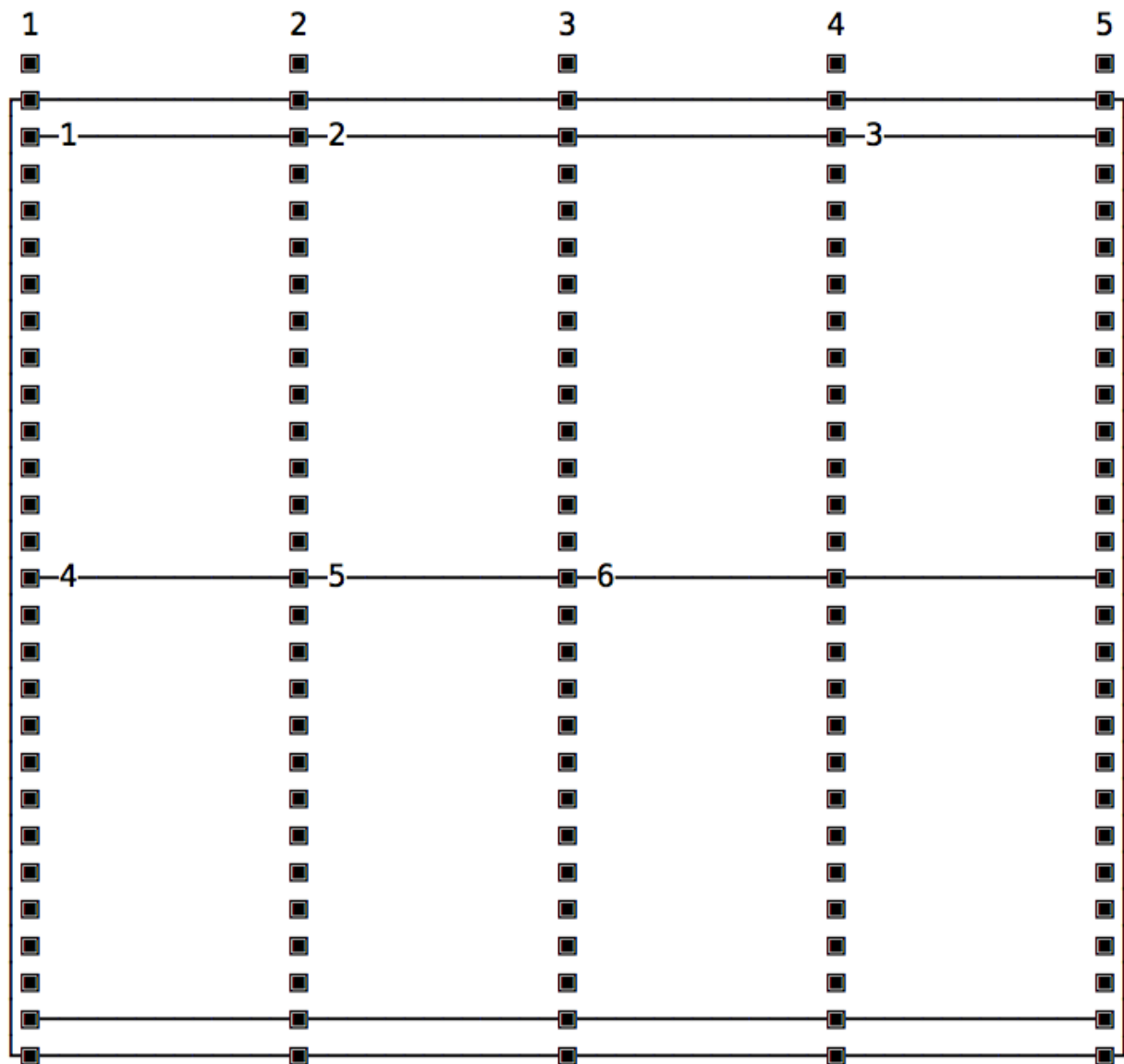
The same principle applies to `grid-row-start` and `grid-row-end`, except this time instead of taking more columns, a cell takes more rows.

Shorthand syntax

Those properties have a shorthand syntax provided by: - `grid-column` - `grid-row`

The usage is simple, here's how to replicate the above layout:

```css
.container {
  display: grid;
  grid-template-columns: 200px 200px 200px 200px;
  grid-template-rows: 300px 300px;
}
.item1 {
  grid-column: 2 / 4;
}
.item6 {
  grid-column: 3 / 5;
}
```

Another approach is to set the starting column/row, and set how many it should occupy using `span`:

```css
.container {
  display: grid;
  grid-template-columns: 200px 200px 200px 200px;
  grid-template-rows: 300px 300px;
}
.item1 {
  grid-column: 2 / span 2;
}
.item6 {
  grid-column: 3 / span 2;
}
```

More grid configuration

Using fractions

Specifying the exact width of each column or row is not ideal in every case.

A fraction is a unit of space.

The following example divides a grid into 3 columns with the same width, ⅓ of the available space each.

```
.container {
  grid-template-columns: 1fr 1fr 1fr;
}
```

Using percentages and rem

You can also use percentages, and mix and match fractions, pixels, rem and percentages:

```
.container {
  grid-template-columns: 3rem 15% 1fr 2fr
}
```

Using `repeat()`

`repeat()` is a special function that takes a number that indicates the number of times a row/column will be repeated, and the length of each one.

If every column has the same width you can specify the layout using this syntax:

```css
.container {
    grid-template-columns: repeat(4, 100px);
}
```

This creates 4 columns with the same width.

Or using fractions:

```css
.container {
    grid-template-columns: repeat(4, 1fr);
}
```

Specify a minimum width for a row

Common use case: Have a sidebar that never collapses more than a certain amount of pixels when you resize the window.

Here's an example where the sidebar takes $\frac{1}{4}$ of the screen and never takes less than 200px:

```css
.container {
    grid-template-columns: minmax(200px, 3fr) 9fr;
}
```

You can also set just a maximum value using the `auto` keyword:

```
.container {
  grid-template-columns: minmax(auto, 50%) 9fr;
}
```

or just a minimum value:

```
.container {
  grid-template-columns: minmax(100px, auto) 9fr;
}
```

Positioning elements using `grid-template-areas`

By default elements are positioned in the grid using their order in the HTML structure.

Using `grid-template-areas` You can define template areas to move them around in the grid, and also to spawn an item on multiple rows / columns instead of using `grid-column`.

Here's an example:

```
<div class="container">
  <main>
    ...
  </main>
```

```
    <aside>
      ...
    </aside>
    <header>
      ...
    </header>
    <footer>
      ...
    </footer>
  </div>
```

```
.container {
  display: grid;
  grid-template-columns: 200px 200px 200px 200px;
  grid-template-rows: 300px 300px;
  grid-template-areas:
    "header header header header"
    "sidebar main main main"
    "footer footer footer footer";
}
main {
  grid-area: main;
}
aside {
  grid-area: sidebar;
}
header {
  grid-area: header;
}
footer {
  grid-area: footer;
}
```

Despite their original order, items are placed where `grid-template-areas` define, depending on the `grid-area` property associated to them.

Adding empty cells in template areas

You can set an empty cell using the dot `.` instead of an area name in `grid-template-areas`:

```css
.container {
  display: grid;
  grid-template-columns: 200px 200px 200px 200px;
  grid-template-rows: 300px 300px;
  grid-template-areas:
    ". header header ."
    "sidebar . main main"
    ". footer footer .";
}
```

Fill a page with a grid

You can make a grid extend to fill the page using `fr`:

```css
.container {
  display: grid;
  height: 100vh;
  grid-template-columns: 1fr 1fr 1fr 1fr;
  grid-template-rows: 1fr 1fr;
}
```

Wrapping up

These are the basics of CSS Grid. There are many things I didn't include in this introduction but I wanted to make it very simple, to start using this new layout system without making it feel overwhelming.

CSS CUSTOM PROPERTIES

Discover CSS Custom Properties, also called CSS Variables, a powerful new feature of modern browsers that help you write better CSS

Introduction

In the last few years CSS preprocessors had a lot of success. It was very common for greenfield projects to start with Less or Sass. And it's still a very popular technology.

The main benefits of those technologies are, in my opinion:

- They allow to nest selectors
- The provide an easy imports functionality
- They give you variables

Modern CSS has a new powerful feature called **CSS Custom Properties**, also commonly known as **CSS Variables**.

CSS is not a programming language like JavaScript, Python, PHP, Ruby or Go where variables are key to do something useful. CSS is very limited in what it can do, and it's mainly a declarative syntax to tell browsers how they should display an HTML page.

But a variable is a variable: a name that refers to a value, and variables in CSS helps reduce repetition and inconsistencies in your CSS, by centralizing the values definition.

And it introduces a unique feature that CSS preprocessors won't never have: **you can access and change the value of a CSS Variable programmatically using JavaScript**.

The basics of using variables

A CSS Variable is defined with a special syntax, prepending **two dashes** to a name (`--variable-name`), then a colon and a value. Like this:

```css
:root {
  --primary-color: yellow;
}
```

(more on `:root` later)

You can access the variable value using `var()`:

```css
p {
  color: var(--primary-color)
}
```

The variable value can be any valid CSS value, for example:

```css
:root {
  --default-padding: 30px 30px 20px 20px;
  --default-color: red;
  --default-background: #fff;
}
```

Create variables inside any element

CSS Variables can be defined inside any element. Some examples:

```css
:root {
  --default-color: red;
}

body {
  --default-color: red;
}

main {
  --default-color: red;
}

p {
  --default-color: red;
}
```

```
span {
  --default-color: red;
}

a:hover {
  --default-color: red;
}
```

What changes in those different examples is the **scope**.

Variables scope

Adding variables to a selector makes them available to all the children of it.

In the example above you saw the use of `:root` when defining a CSS variable:

```
:root {
  --primary-color: yellow;
}
```

`:root` is a CSS pseudo-class that identifies the document, so adding a variable to `:root` makes it available to all the elements in the page.

It's just like targeting the `html` element, except that `:root` has higher specificity (takes priority).

If you add a variable inside a `.container` selector, it's only going to be available to children of `.container`:

```css
.container {
  --secondary-color: yellow;
}
```

and using it outside of this element is not going to work.

Variables can be **reassigned**:

```css
:root {
  --primary-color: yellow;
}

.container {
  --primary-color: blue;
}
```

Outside `.container`, `--primary-color` will be *yellow*, but inside it will be *blue*.

You can also assign or overwrite a variable inside the HTML using **inline styles**:

```html
<main style="--primary-color: orange;">
  <!-- ... -->
</main>
```

> *CSS Variables follow the normal CSS cascading rules, with precedence set according to specificity*

Interacting with a CSS Variable value using JavaScript

The coolest thing with CSS Variables is the ability to access and edit them using JavaScript.

Here's how you set a variable value using plain JavaScript:

```
const element = document.getElementById('my-element')
element.style.setProperty('--variable-name', 'a-value')
```

This code below can be used to access a variable value instead, in case the variable is defined on `:root`:

```
const styles = getComputedStyle(document.documentElement)
const value = String(styles.getPropertyValue('--variable-name')).trim()
```

Or, to get the style applied to a specific element, in case of variables set with a different scope:

```
const element = document.getElementById('my-element')
const styles = getComputedStyle(element)
```

```
const value = String(styles.getPropertyValue('--variable-
name')).trim()
```

Handling invalid values

If a variable is assigned to a property which does not accept the variable value, it's considered invalid.

For example you might pass a pixel value to a `position` property, or a rem value to a color property.

In this case the line is considered invalid and ignored.

Browser support

Browser support for CSS Variables is **very good**, according to Can I Use (https://www.caniuse.com/#feat=css-variables) .

CSS Variables are here to stay, and you can use them today if you don't need to support Internet Explorer and old versions of the other browsers.

If you need to support older browsers you can use libraries like PostCSS or Myth (http://www.myth.io/) , but you'll lose the ability to interact with variables via JavaScript or the Browser Developer Tools,

as they are transpiled to good old variable-less CSS (and as such, you lose most of the power of CSS Variables).

CSS Variables are case sensitive

This variable:

```
--width: 100px;
```

is different than:

```
--Width: 100px;
```

Math in CSS Variables

To do math in CSS Variables, you need to use `calc()`, for example:

```
:root {
  --default-left-padding: calc(10px * 2);
}
```

Media queries with CSS Variables

Nothing special here. CSS Variables normally apply to media queries:

```css
body {
    --width: 500px;
}

@media screen and (max-width: 1000px) and (min-width: 700px) {
    --width: 800px;
}

.container {
  width: var(--width);
}
```

Setting a fallback value for var()

`var()` accepts a second parameter, which is the default fallback value when the variable value is not set:

```css
.container {
  margin: var(--default-margin, 30px);
}
```

POSTCSS

Discover PostCSS, a great tool to help you write modern CSS. PostCSS is a very popular tool that allows developers to write CSS pre-processors or post-processors

- Introduction

Introduction

PostCSS is a very popular tool that allows developers to write CSS pre-processors or post-processors, called **plugins**. There is a huge number of plugins that provide lots of functionalities, and sometimes the term "PostCSS" means the tool itself, plus the plugins ecosystem.

PostCSS plugins can be run via the command line, but they are generally invoked by task runners at build time.

The plugin-based architecture provides a common ground for all the CSS-related operations you need.

Note: PostCSS despite the name is not a CSS post-processor, but it can be used to build them, as well as other things

Why it's so popular

PostCSS provides several features that will *deeply improve your CSS*, and it integrates really well with any build tool of your choice.

Install the PostCSS CLI

Using Yarn:

```
yarn global add postcss-cli
```

or npm:

```
npm install -g postcss-cli
```

Once this is done, the `postcss` command will be available in your command line.

This command for example runs the autoprefixer plugin on CSS files contained in the css/ folder, and save the result in the main.css file:

```
postcss --use autoprefixer -o main.css css/*.css
```

More info on the PostCSS CLI here:
https://github.com/postcss/postcss-cli.

Most popular PostCSS plugins

PostCSS provides a common interface to several great tools for your CSS processing.

Here are some of the most popular plugins, to get an overview of what's possible to do with PostCSS.

Autoprefixer

Autoprefixer (https://github.com/postcss/autoprefixer) parses your CSS and determines if some rules need a vendor prefix.

It does so according to the Can I Use (http://caniuse.com/) data, so you don't have to worry if a feature needs a prefix, or if prefixes you use are now unneeded because obsolete.

You get to write cleaner CSS.

Example:

```css
a {
    display: flex;
}
```

gets compiled to

```css
a {
    display: -webkit-box;
    display: -webkit-flex;
    display: -ms-flexbox;
    display: flex;
}
```

cssnext

https://github.com/MoOx/postcss-cssnext

This plugin is the Babel of CSS, allows you to use modern CSS features while it takes care of transpiling them to a CSS more digestible to older browsers:

- it adds prefixes using Autoprefixer (so if you use this, no need to use Autoprefixer directly)
- it allows you to use CSS Variables
- it allows you to use nesting, like in Sass

and a lot more (http://cssnext.io/features/) !

CSS Modules

CSS Modules (https://github.com/css-modules/postcss-modules) let you use CSS Modules.

CSS Modules are not part of the CSS standard, but they are a build step process to have scoped selectors.

csslint

Linting helps you write correct CSS and avoid errors or pitfalls. The stylint (http://stylelint.io/) plugin allows you to lint CSS at build time.

cssnano

cssnano (http://cssnano.co/) minifies your CSS and makes code optimizations to have the least amount of code delivered in production.

Other useful plugins

On the PostCSS GitHub repo there is a full list of the available plugins (https://github.com/postcss/postcss/blob/master/docs/plugins.md) .

Some of the ones I like include:

- LostGrid (https://github.com/peterramsing/lost) is a PostCSS grid system
- postcss-sassy (https://github.com/andyjansson/postcss-sassy-mixins) provides Sass-like mixins
- postcss-nested (https://github.com/postcss/postcss-nested) provides the ability to use Sass nested rules
- postcss-nested-ancestors (https://github.com/toomuchdesign/postcss-nested-ancestors) , reference any ancestor selector in nested CSS
- postcss-simple-vars (https://github.com/postcss/postcss-simple-vars) , use Sass-like variables
- PreCSS (https://github.com/jonathantneal/precss) provides you many features of Sass, and this is what is most close to a complete Sass replacement

How is it different than Sass?

Or any other CSS preprocessor?

The main benefit PostCSS provides over CSS preprocessors like Sass or Less is the ability to choose your own path, and cherry-pick the features you need, adding new capabilities at the same time. Sass or Less are "fixed", you get lots of features which you might or might not use, and you cannot extend them.

The fact that you "choose your own adventure" means that you can still use any other tool you like alongside PostCSS. You can still use Sass if this is what you want, and use PostCSS to perform other things that Sass can't do, like autoprefixing or linting.

You can write your own PostCSS plugin to do anything you want.

CSS TRANSITIONS

CSS Transitions are the most simple way to create an animation in CSS. In a transition, you change the value of a property, and you tell CSS to slowly change it according to some parameters, towards a final state

- Introduction to CSS Transitions
- Example of a CSS Transition
- Transition timing function values
- CSS Transitions in Browser DevTools
- Which Properties you can Animate using CSS Animations

Introduction to CSS Transitions

CSS Transitions are the most simple way to create an animation in CSS.

In a transition, you change the value of a property, and you tell CSS to slowly change it according to some parameters, towards a final state.

CSS Transitions are defined by these properties:

Property	Description
`transition-property`	the CSS property that should transition
`transition-duration`	the duration of the transition
`transition-timing-function`	the timing function used by the animation (common values: linear, ease). Default: ease
`transition-delay`	optional number of seconds to wait before starting the animation

The `transition` property is a handy shorthand:

```
.container {
  transition: property
              duration
              timing-function
```

```
        delay;
}
```

Example of a CSS Transition

This code implements a CSS Transition:

```css
.one,
.three {
  background: rgba(142, 92, 205, .75);
  transition: background 1s ease-in;
}

.two,
.four {
  background: rgba(236, 252, 100, .75);
}

.circle:hover {
  background: rgba(142, 92, 205, .25); /* lighter */
}
```

See the example on Glitch https://flavio-css-transitions-example.glitch.me

When hovering the `.one` and `.three` elements, the purple circles, there is a transition animation that ease the change of background, while the yellow circles do not, because they do not have the `transition` property defined.

Transition timing function values

`transition-timing-function` allows to specify the acceleration curve of the transition.

There are some simple values you can use:

Value

linear

ease

ease-in

ease-out

ease-in-out

This Glitch (https://flavio-css-transitions-easings.glitch.me) shows how those work in practice.

You can create a completely custom timing function using cubic bezier curves (https://developer.mozilla.org/en-US/docs/Web/CSS/single-transition-timing-function) . This is rather

advanced, but basically any of those functions above is built using bezier curves. We have handy names as they are common ones.

CSS Transitions in Browser DevTools

The Browser DevTools offer a great way to visualize transitions.

This is Chrome:
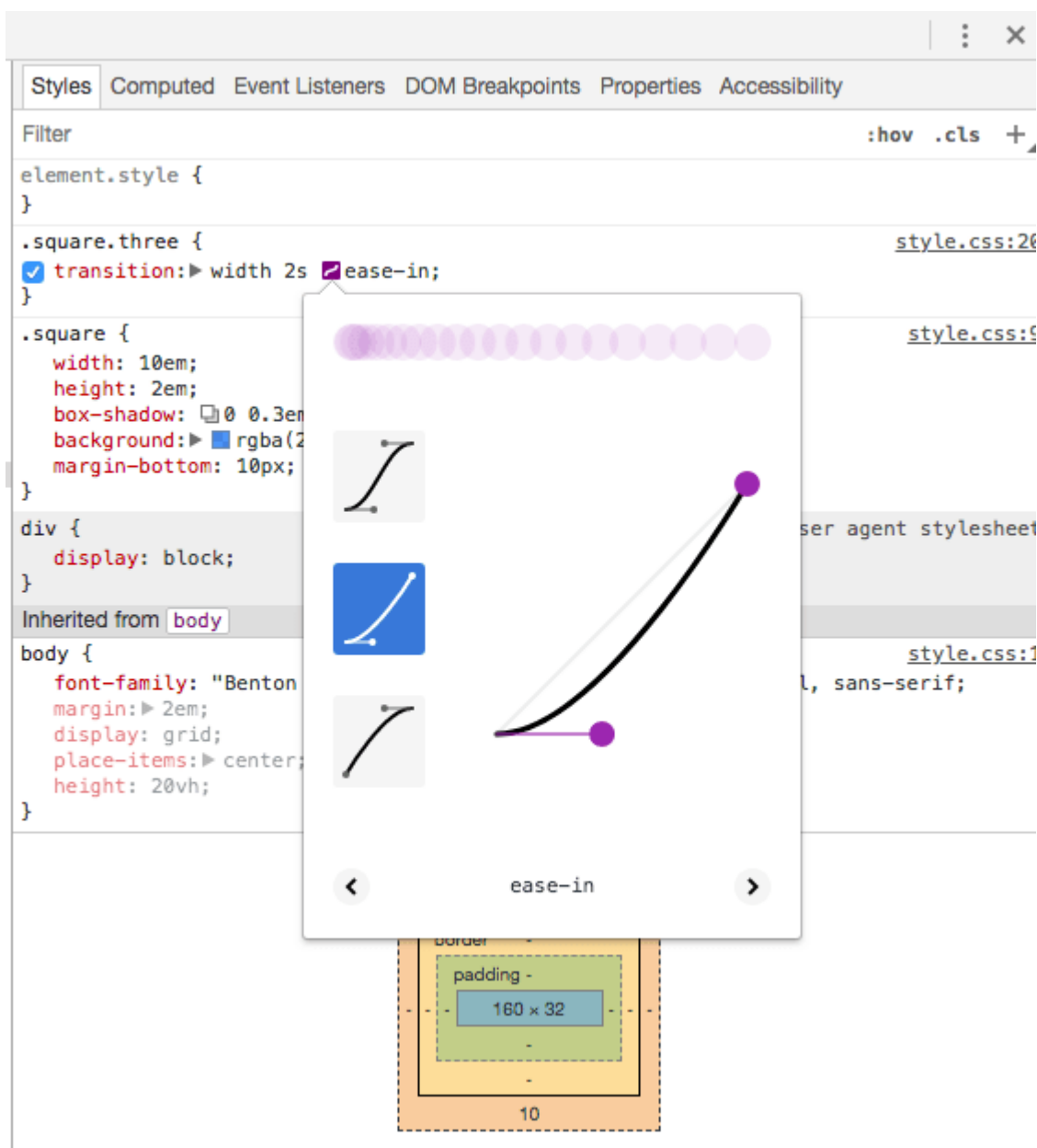
This is Firefox:
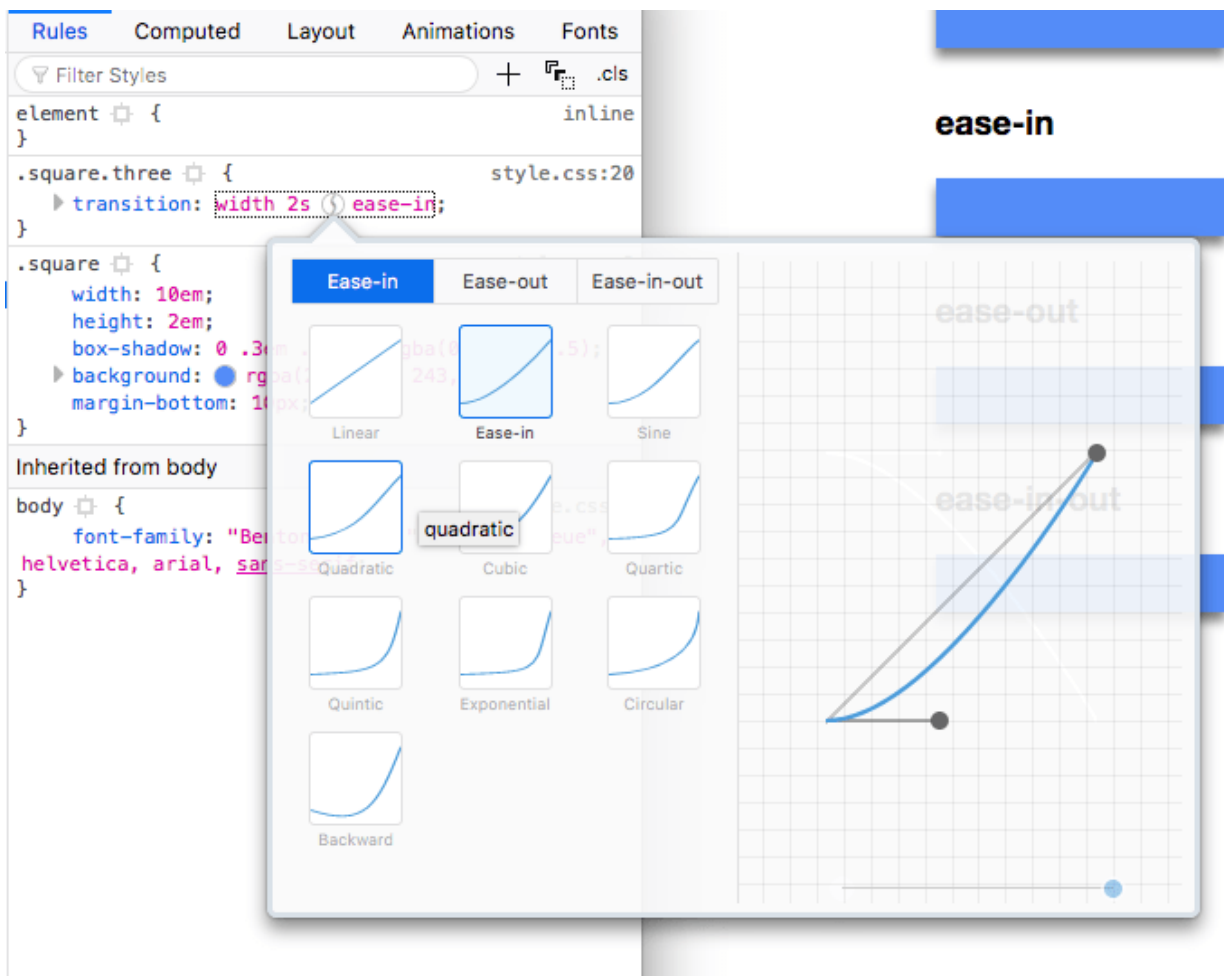
From those panels you can live edit the transition and experiment in the page directly without reloading your code.

Which Properties you can Animate using CSS Animations

A lot! They are the same you can animate using CSS Transitions, too.

Here's the full list:

Property

background

background-color

background-position

background-size

border

border-color

border-width

border-bottom

border-bottom-color

border-bottom-left-radius

border-bottom-right-radius

border-bottom-width

border-left

border-left-color

border-left-width

border-radius

border-right

border-right-color

border-right-width

border-spacing

Property

border-top

border-top-color

border-top-left-radius

border-top-right-radius

border-top-width

bottom

box-shadow

caret-color

clip

color

column-count

column-gap

column-rule

column-rule-color

column-rule-width

column-width

columns

content

filter

flex

Property

flex-basis

flex-grow

flex-shrink

font

font-size

font-size-adjust

font-stretch

font-weight

grid-area

grid-auto-columns

grid-auto-flow

grid-auto-rows

grid-column-end

grid-column-gap

grid-column-start

grid-column

grid-gap

grid-row-end

grid-row-gap

grid-row-start

Property

grid-row

grid-template-areas

grid-template-columns

grid-template-rows

grid-template

grid

height

left

letter-spacing

line-height

margin

margin-bottom

margin-left

margin-right

margin-top

max-height

max-width

min-height

min-width

opacity

Property

order

outline

outline-color

outline-offset

outline-width

padding

padding-bottom

padding-left

padding-right

padding-top

perspective

perspective-origin

quotes

right

tab-size

text-decoration

text-decoration-color

text-indent

text-shadow

top

Property

transform.

vertical-align

visibility

width

word-spacing

z-index

CSS ANIMATIONS

CSS Animations are a great way to create visual animations, not limited to a single movement like CSS Transitions, but much more articulated. An animation is applied to an element using the `animation` property

- Introduction
- A CSS Animations Example
- The CSS animation properties

- JavaScript events for CSS Animations
- Which Properties You Can Animate using CSS Animations

Introduction

An animation is applied to an element using the `animation` property.

```css
.container {
    animation: spin 10s linear infinite;
}
```

`spin` is the name of the animation, which we need to define separately. We also tell CSS to make the animation last 10 seconds, perform it in a linear way (no acceleration or any difference in its speed) and to repeat it infinitely.

You must **define how your animation works** using **keyframes**. Example of an animation that rotates an item:

```css
@keyframes spin {
    0% {
        transform: rotateZ(0);
    }
    100% {
        transform: rotateZ(360deg);
    }
}
```

Inside the `@keyframes` definition you can have as many intermediate waypoints as you want.

In this case we instruct CSS to make the transform property to rotate the Z axis from 0 to 360 grades, completing the full loop.

You can use any CSS transform here.

Notice how this does not dictate anything about the temporal interval the animation should take. This is defined when you use it via `animation`.

A CSS Animations Example

I want to draw four circles, all with a starting point in common, all 90 degrees distant from each other.

```
<div class="container">
  <div class="circle one"></div>
  <div class="circle two"></div>
  <div class="circle three"></div>
  <div class="circle four"></div>
</div>
```

```
body {
  display: grid;
  place-items: center;
  height: 100vh;
}
```

```css
.circle {
    border-radius: 50%;
    left: calc(50% - 6.25em);
    top: calc(50% - 12.5em);
    transform-origin: 50% 12.5em;
    width: 12.5em;
    height: 12.5em;
    position: absolute;
    box-shadow: 0 1em 2em rgba(0, 0, 0, .5);
}

.one,
.three {
    background: rgba(142, 92, 205, .75);
}

.two,
.four {
    background: rgba(236, 252, 100, .75);
}


.one {
    transform: rotateZ(0);
}

.two {
    transform: rotateZ(90deg);
}

.three {
    transform: rotateZ(180deg);
}

.four {
    transform: rotateZ(-90deg);
}
```

You can see them in this Glitch: https://flavio-css-circles.glitch.me

Let's make this structure (all the circles together) rotate. To do this, we apply an animation on the container, and we define that animation as a 360 degrees rotation:

```css
@keyframes spin {
    0% {
        transform: rotateZ(0);
    }
    100% {
        transform: rotateZ(360deg);
    }
}

.container {
    animation: spin 10s linear infinite;
}
```

See it on https://flavio-css-animations-tutorial.glitch.me

You can add more keyframes to have funnier animations:

```css
@keyframes spin {
    0% {
        transform: rotateZ(0);
    }
    25% {
```

```
        transform: rotateZ(30deg);
    }
    50% {
        transform: rotateZ(270deg);
    }
    75% {
        transform: rotateZ(180deg);
    }
    100% {
        transform: rotateZ(360deg);
    }
}
```

See the example on https://flavio-css-animations-four-steps.glitch.me

The CSS animation properties

CSS animations offers a lot of different parameters you can tweak:

Property	Description
`animation-name`	the name of the animation, it references an animation created using `@keyframes`
`animation-duration`	how long the animation should last, in seconds
`animation-timing-function`	the timing function used by the animation (common values: `linear`, `ease`). Default: `ease`

Property	Description
animation-delay	optional number of seconds to wait before starting the animation
animation-iteration-count	how many times the animation should be performed. Expects a number, or `infinite`. Default: 1
animation-direction	the direction of the animation. Can be `normal`, `reverse`, `alternate` or `alternate-reverse`. In the last 2, it alternates going forward and then backwards
animation-fill-mode	defines how to style the element when the animation ends, after it finishes its iteration count number. `none` or `backwards` go back to the first keyframe styles. `forwards` and `both` use the style that's set in the last keyframe
animation-play-state	if set to `paused`, it pauses the animation. Default is `running`

The `animation` property is a shorthand for all these properties, in this order:

```css
.container {
  animation: name
             duration
             timing-function
             delay
             iteration-count
             direction
             fill-mode
             play-state;
}
```

This is the example we used above:

```css
.container {
    animation: spin 10s linear infinite;
}
```

JavaScript events for CSS Animations

Using JavaScript you can listen for the following events:

- `animationstart`
- `animationend`
- `animationiteration`

Be careful with `animationstart`, because if the animation starts on page load, your JavaScript code is always executed after the CSS has been processed, so the animation is already started and you cannot intercept the event.

```javascript
const container = document.querySelector('.container')
container.addEventListener('animationstart', (e) => {

    //do something
}, false)
container.addEventListener('animationend', (e) => {
    //do something
}, false)
container.addEventListener('animationiteration', (e) => {
    //do something
}, false)
```

Which Properties You Can Animate using CSS Animations

A lot! They are the same you can animate using CSS Transitions, too.

Here's the full list:

Property

background

background-color

background-position

background-size

border

border-color

border-width

border-bottom

border-bottom-color

border-bottom-left-radius

border-bottom-right-radius

border-bottom-width

border-left

border-left-color

Property

border-left-width

border-radius

border-right

border-right-color

border-right-width

border-spacing

border-top

border-top-color

border-top-left-radius

border-top-right-radius

border-top-width

bottom

box-shadow

caret-color

clip

color

column-count

column-gap

column-rule

column-rule-color

Property

column-rule-width

column-width

columns

content

filter

flex

flex-basis

flex-grow

flex-shrink

font

font-size

font-size-adjust

font-stretch

font-weight

grid-area

grid-auto-columns

grid-auto-flow

grid-auto-rows

grid-column-end

grid-column-gap

Property

grid-column-start

grid-column

grid-gap

grid-row-end

grid-row-gap

grid-row-start

grid-row

grid-template-areas

grid-template-columns

grid-template-rows

grid-template

grid

height

left

letter-spacing

line-height

margin

margin-bottom

margin-left

margin-right

Property

margin-top

max-height

max-width

min-height

min-width

opacity

order

outline

outline-color

outline-offset

outline-width

padding

padding-bottom

padding-left

padding-right

padding-top

perspective

perspective-origin

quotes

right

Property

tab-size

text-decoration

text-decoration-color

text-indent

text-shadow

top

transform.

vertical-align

visibility

width

word-spacing

z-index

HOW TO CENTER THINGS WITH CSS

Centering elements with CSS has always been easy for some things, hard for others. Here is the full list of centering techniques, with modern CSS techniques as well

Centering things in CSS is a task that is very different if you need to center horizontally or vertically.

In this post I explain the most common scenarios and how to solve them. If a new solution is provided by Flexbox I ignore the old techniques because we need to move forward, and Flexbox is supported by browsers since years, IE10 included.

Center horizontally

Text

Text is very simple to center horizontally using the `text-align` property set to `center`:

```css
p {
    text-align: center;
}
```

Blocks

The modern way to center anything that is not text is to use Flexbox:

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

any element inside `#mysection` will be horizontally centered.
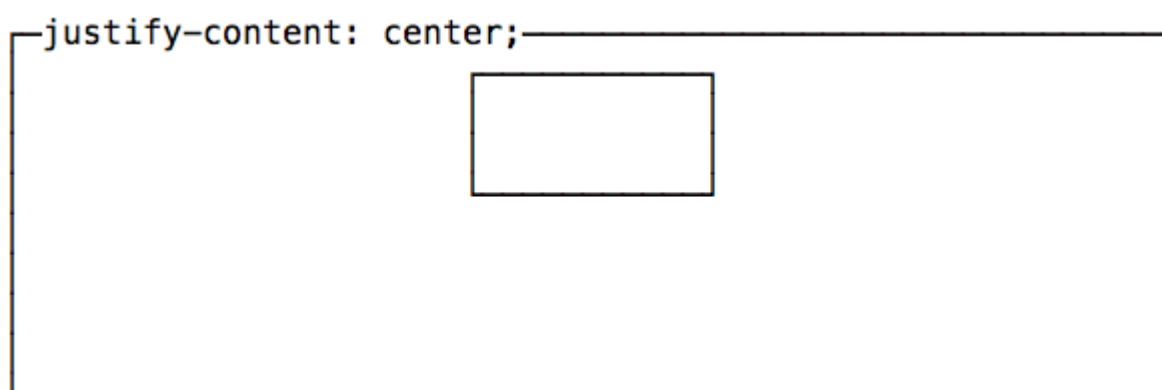
Here is the alternative approach if you don't want to use Flexbox.

Anything that is not text can be centered by applying an automatic margin to left and right, and setting the width of the element:

```css
section {
  margin-left: 0 auto;
  width: 50%;
}
```

the above `margin-left: 0 auto;` is a shorthand for:

```css
section {
  margin-top: 0;
  margin-bottom: 0;
  margin-left: auto;
  margin-right: auto;
}
```

Remember to set the item to `display: block` if it's an inline element.

Center vertically

Traditionally this has always been a difficult task. Flexbox now provides us a great way to do this in the simplest possible way:

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

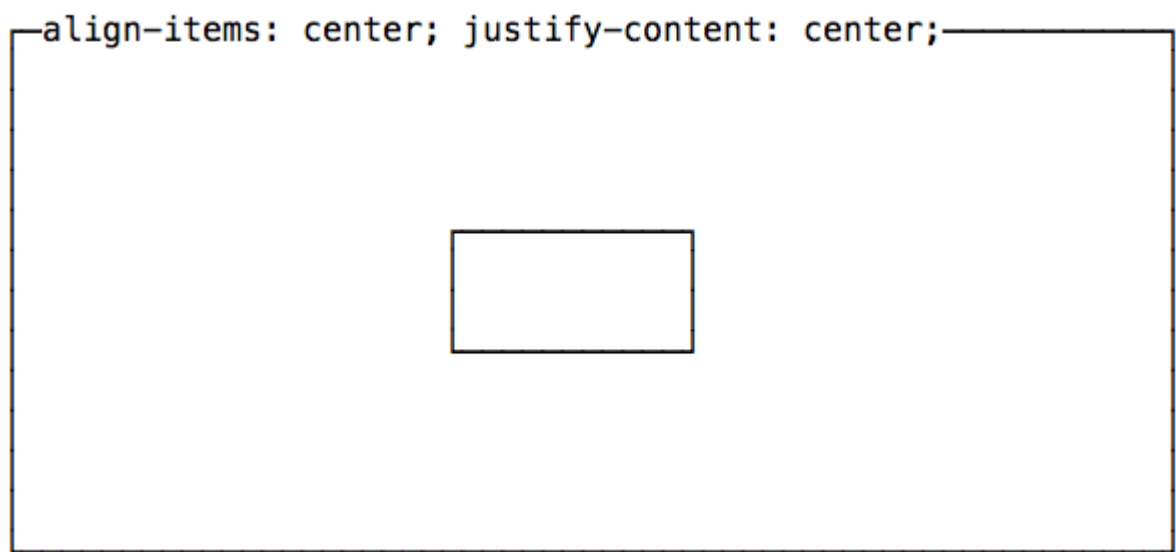any element inside `#mysection` will be vertically centered.
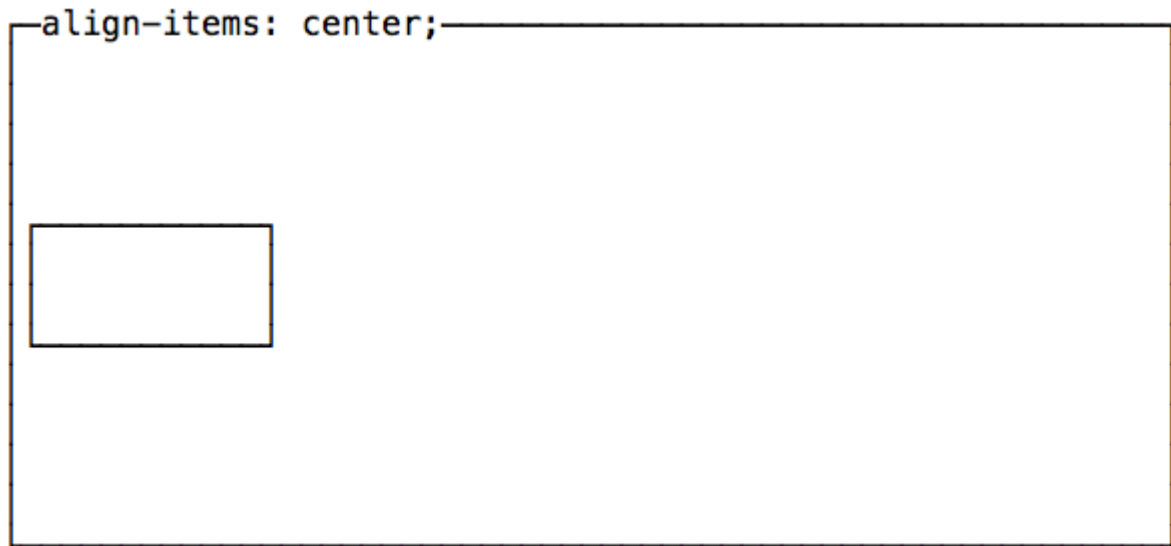
Center both vertically and horizontally

Flexbox techniques to center vertically and horizontally can be combined to completely center an element in the page.

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

```
┌─align-items: center; justify-content: center;──────────┐
│                                                         │
│                                                         │
│                                                         │
│                      ┌──────────┐                       │
│                      │          │                       │
│                      │          │                       │
│                      └──────────┘                       │
│                                                         │
│                                                         │
│                                                         │
└─────────────────────────────────────────────────────────┘
```

The same can be done using CSS Grid:

```css
body {
  display: grid;
  place-items: center;
  height: 100vh;
}
```

THE CSS MARGIN PROPERTY

margin is a simple CSS property that has a shorthand syntax I keep forgetting about, so I wrote this reference post

- Introduction
- Specific margin properties
- Using `margin` with different values

  - 1 value
  - 2 values
  - 3 values
  - 4 values

- Values accepted
- Using `auto` to center elements

Introduction

The `margin` CSS property is commonly used in CSS to add space around an element.

Remember:

- `margin` adds space around an element border
- `padding` adds space inside an element border

Specific margin properties

`margin` has 4 related properties that alter the margin of a single margin at once:

- `margin-top`
- `margin-right`
- `margin-bottom`
- `margin-left`

The usage of those is very simple and cannot be confused, for example:

```
margin-left: 30px;
margin-right: 3em;
```

Using `margin` with different values

`margin` is a shorthand to specify multiple margins at the same time, and depending on the number of values entered, it behaves differently.

1 value

Using a single value applies that to **all** the margins: top, right, bottom, left.

```
margin: 20px;
```

2 values

Using 2 values applies the first to **bottom & top**, and the second to **left & right**.

```
margin: 20px 10px;
```

3 values

Using 3 values applies the first to **top**, the second to **left & right**, the third to **bottom**.

```
margin: 20px 10px 30px;
```

4 values

Using 4 values applies the first to **top**, the second to **right**, the third to **bottom**, the fourth to **left**.

```
margin: 20px 10px 5px 0px;
```

So, the order is *top-right-bottom-left*.

Values accepted

`margin` accepts values expressed in any kind of length unit, the most common ones are px, em, rem, but many others exist (https://developer.mozilla.org/en-US/docs/Web/CSS/length) .

It also accepts percentage values, and the special value `auto`.

Using `auto` to center elements

`auto` can be used to tell the browser to select automatically a margin, and it's most commonly used to center an element in this way:

```
margin: 0 auto;
```

As said above, using 2 values applies the first to **bottom & top**, and the second to **left & right**.

The modern way to center elements is to use Flexbox, and its `justify-content: center;` directive.

Older browsers of course do not implement Flexbox, and if you need to support them `margin: 0 auto;` is still a good choice.

CSS SYSTEM FONTS

How to use System Fonts in CSS to improve your site and provide a better experience to your users in terms of speed and page load time

- A little bit of history

- Today

- The impact of Web Fonts

- Enter System Fonts
- Popular websites use System Fonts
- I'm sold. Give me the code
  - A note on `system-ui`
- Use font variations by creating @font-face rules
- Read more

A little bit of history

For *years*, websites could only use fonts available on all computers, such as Georgia, Verdana, Arial, Helvetica, Times New Roman. Other fonts were not guaranteed to work on all websites.

If you wanted to use a fancy font you had to use images.

In 2008 Safari and Firefox introduced the `@font-face` CSS property, and online services started to provide licenses to Web Fonts. The first was Typekit in 2009, and later Google Fonts got hugely popular thanks to its free offering.

`@font-face` was implemented in all the major browsers, and nowadays it's a given on every reasonably recent device. If you're a young web developer you might not realize it, but in 2012 we still had articles explaining this new technology of Web Fonts.

Today

You *can* use whatever font you wish to use, by relying on a service like Google Fonts, or providing your own font to download.

You *can*, but **should you**?

If you have the choice (and by this I mean, you're not implementing a design that a client gave you), you might want to think about it, in a move to go back to the basics (but in style!)

The impact of Web Fonts

Everything you load on your pages has a **cost**. This cost is especially impactful on mobile, where every byte you require is impacting the load time, and the amount of bandwidth you make your users consume.

The font must load before the content renders, so you need to **wait** for that resource loading to complete before the user is able to read even a single word you wrote.

But Web Fonts are a way to provide an **awesome user experience** through good typography.

Enter System Fonts

Operating Systems have great default fonts.

System Fonts offer the great advantage of **speed** and **performance**, and a **reduction of your web page size**.

But as a side effect, they make your website look **very familiar** to anyone looking at it, because they are used to see that same font every day on their computer or mobile device.

It's effectively a **native font**.

And as it's the system font, it's **guaranteed to look great**.

Popular websites use System Fonts

You might know one of these, as an example:

- GitHub
- Medium
- Ghost
- Bootstrap
- Booking.com

..they have been using System Fonts for years.

Even the Wordpress dashboard - that runs millions of websites - uses system fonts, and Medium, which is all about reading, decided to use system fonts.

If it works for them, chances are it works for you as well.

I'm sold. Give me the code

This is the CSS line you should add to your website:

```css
body {
  font-family: -apple-system, BlinkMacSystemFont, "Segoe UI",
  "Roboto", "Oxygen", "Ubuntu", "Helvetica Neue", Arial, sans-
  serif;
}
```

The browser will interpret all these font names, and starting from the first it will check if it's available.

Safari and Firefox on macOS "intercept" `-apple-system`, which means the San Francisco font on newer versions, Helvetica Neue and Lucida Grande on older versions.

Chrome works with BlinkMacSystemFont, which defaults to the OS font (again, San Francisco on macOS).

Segoe UI is used in modern Windows systems and Windows Phone, Tahoma in Windows XP, Roboto in Android, and so on targeting

other platforms.

Arial and sans-serif are the fallback fonts.

If you use **Emojis** in your site, make sure you load the symbol fonts as well:

```css
body {
  font-family: -apple-system, BlinkMacSystemFont, "Segoe UI",
"Roboto",
  "Oxygen", "Ubuntu", "Helvetica Neue", Arial, sans-serif,
"Apple Color Emoji",
  "Segoe UI Emoji", "Segoe UI Symbol";
}
```

You might want to change the order of the font appearance based on your website usage statistics.

A note on `system-ui`

Maybe you will see `system-ui` mentioned in System Fonts posts online, but at the moment it's known to cause issues in Windows (see https://infinnie.github.io/blog/2017/systemui.html and https://github.com/twbs/bootstrap/pull/22377)

There is work being done towards standardizing system-ui as a generic font family, so in the future you will just write

```css
body {
  font-family: system-ui;
}
```

See https://www.chromestatus.com/feature/5640395337760768 and https://caniuse.com/#feat=font-family-system-ui to keep an eye on the progress. Chrome, Safari already support it, Firefox partially, while Edge does not yet implement it.

Use font variations by creating @font-face rules

The approach described above works *great* until you need to alter the font on a second element, and maybe even on more than one.

Maybe you want to specify the italic as a `font` property rather than in `font-style`, or set a specific font weight.

This nice project by Jonathan Neal https://jonathantneal.github.io/system-font-css/ lets you use System Fonts by simply importing a module, and you can set

```css
body {
  font-family: system-ui;
}
```

This `system-ui` is defined in https://github.com/jonathantneal/system-font-css/blob/gh-pages/system-font.css.

You are now able to use different font variations by referencing:

```css
.special-text {
  font: italic 300 system-ui;
}

p {
  font: 400 system-ui;
}
```

Read more

- https://css-tricks.com/snippets/css/system-font-stack/
- https://www.smashingmagazine.com/2015/11/using-system-ui-fonts-practical-guide/
- https://medium.design/system-shock-6b1dc6d6596f

CSS FOR PRINT

A few tips on printing from the browser to the printer
or to a PDF document using CSS

- Print CSS
- CSS @media print
- Links
- Page margins
- Page breaks
- Avoid breaking images in the middle
- PDF Size
- Debug the printing presentation

Even though we increasingly stare at our screens, printing is still a thing.

Even with blog posts. I remember one time back in 2009 I met a person that told me he made his personal assistant print every blog post I published (yes, I stared blankly for a little bit). Definitely unexpected.

My main use case for looking into printing usually is printing to a PDF. I might create something inside the browser, and I want to make it available as PDF.

Browsers make this very easy, with Chrome defaulting to "Save" when trying to print a document and a printer is not available, and Safari has a dedicated button in the menu bar:
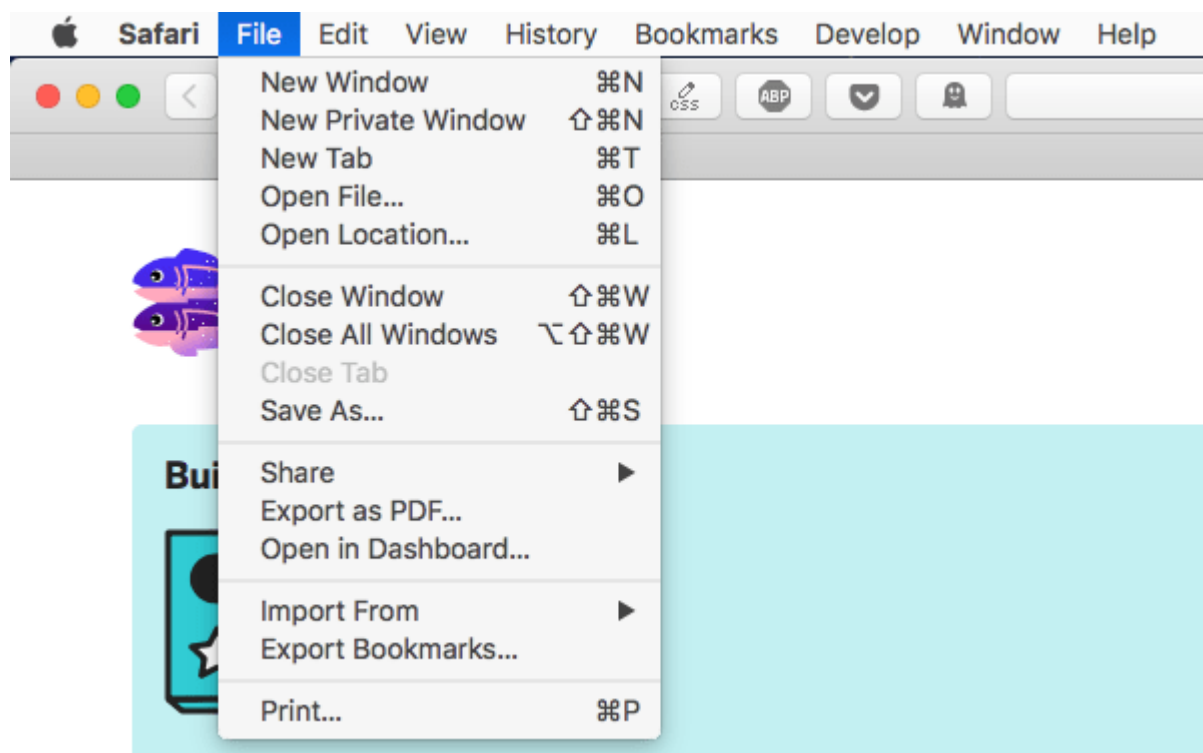
Print CSS

Some common things you might want to do when printing is to hide some parts of the document, maybe the footer, something in the header, the sidebar.

Maybe you want to use a different font for printing, which is totally legit.

If you have a large CSS for print, you'd better use a separate file for it. Browsers will only download it when printing:

```html
<link rel="stylesheet"
      src="print.css"
      type="text/css"
      media="print" />
```

CSS @media print

An alternative to the previous approach is media queries. Anything you add inside this block:

```css
@media print {
  /* ... */
}
```

is going to be applied only to printed documents.

Links

HTML is great because of links. It's called HyperText for a good reason. When printing we might lose a lot of information, depending on the content.

CSS offers a great way to solve this problem by editing the content, appending the link after the `<a>` tag text, using:

```
@media print {
    a[href*='//']:after {
        content:" (" attr(href) ") ";
        color: $primary;
    }
}
```

I target `a[href*='//']` to only do this for external links. I might have internal links for navigation and internal indexing purposes, which would be useless in most of my use cases. If you also want internal links to be printed, just do:

```
@media print {
    a:after {
        content:" (" attr(href) ") ";
        color: $primary;
    }
}
```

Page margins

You can add margins to every single page. `cm` or `in` is a good unit for paper printing.

```css
@page {
    margin-top: 2cm;
    margin-bottom: 2cm;
    margin-left: 2cm;
    margin-right: 2cm;
}
```

`@page` can also be used to only target the first page, using `@page :first`, or only the left and right pages using `@page :left` and `@page: right`.

Page breaks

You might want to add a page break after some elements, or before them. Use `page-break-after` and `page-break-before`:

```css
.book-date {
    page-break-after: always;
}

.post-content {
    page-break-before: always;
}
```

Those properties accept a wide variety of values
[(https://developer.mozilla.org/en-US/docs/Web/CSS/page-break-after)](https://developer.mozilla.org/en-US/docs/Web/CSS/page-break-after) .

Avoid breaking images in the middle

I experienced this with Firefox: images by default are cut in the middle, and continue on the next page. It might also happen to text.

Use

```
p {
    page-break-inside: avoid;
}
```

and wrap your images in a `p` tag. Targeting `img` directly didn't work in my tests.

This applies to other content as well, not just images. If you notice something is cut when you don't want, use this property.

PDF Size

Trying to print a 400+ pages PDF with images with Chrome initially generated a 100MB+ file, although the total size of the images was not nearly that big.

I tried with Firefox and Safari, and the size was less than 10MB.

After a few experiments it turned out Chrome has 3 ways to print an HTML to PDF:

- ❌ Don't print it using the System Dialogue
- ❌ Don't click "Open PDF in Preview"
- ✅ Instead, click the "Save" button that appears in the Chrome Print dialogue

This generates a PDF much quicker than with the other 2 ways, and with a much, much smaller size.

Debug the printing presentation

The Chrome DevTools offer ways to emulate the print layout:
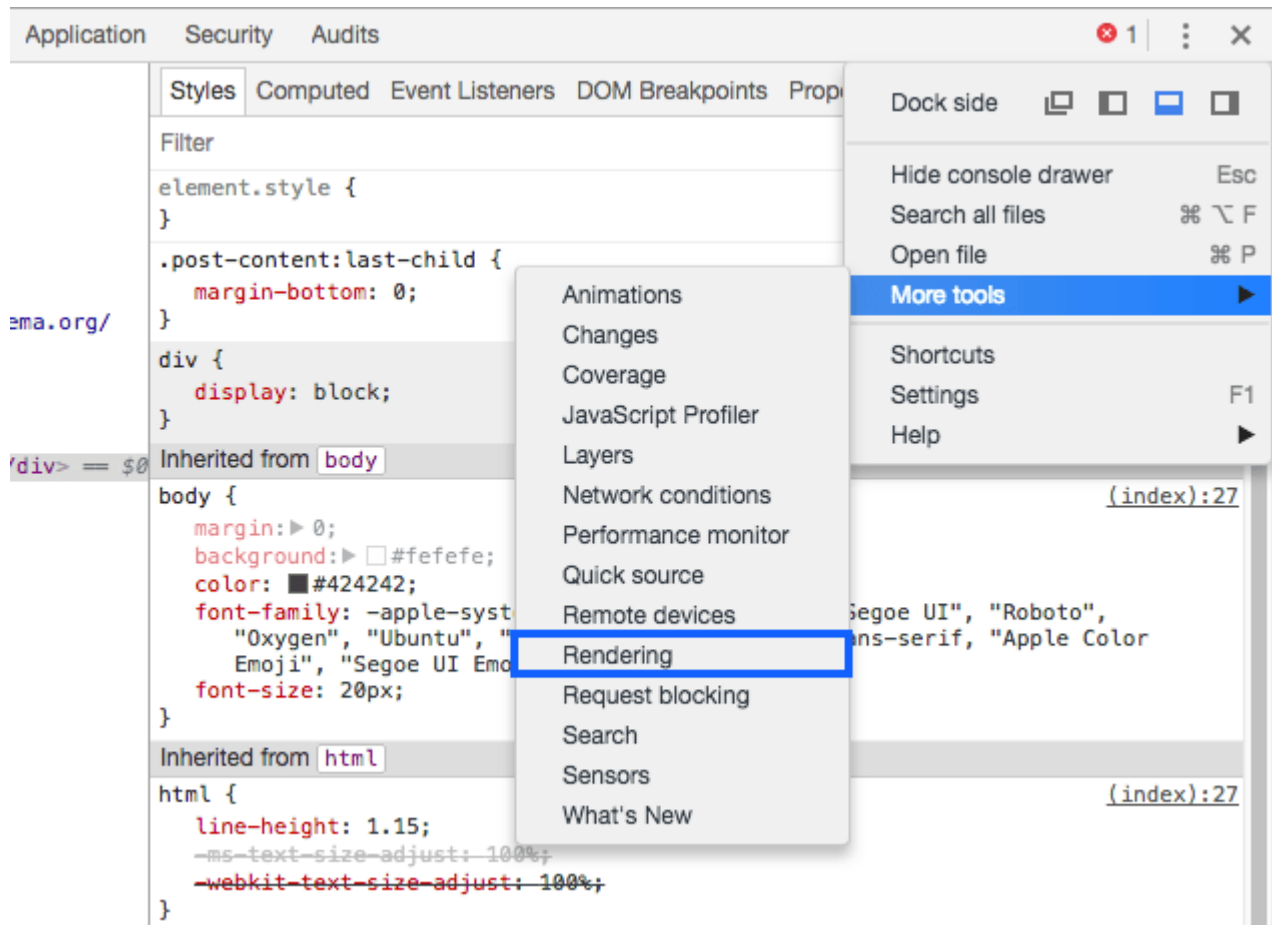
Once the panel opens, change the rendering emulation to `print`:

Paint flashing
Highlights areas of the page (green) that need to be repainted

Layer borders
Shows layer borders (orange/olive) and tiles (cyan)

FPS meter
Plots frames per second, frame rate distribution, and GPU memory

Scrolling performance issues
Highlights elements (teal) that can slow down scrolling, including touch & wheel event handlers

Emulate CSS media
Forces media type for testing print and screen styles

✓ No emulation
print
screen