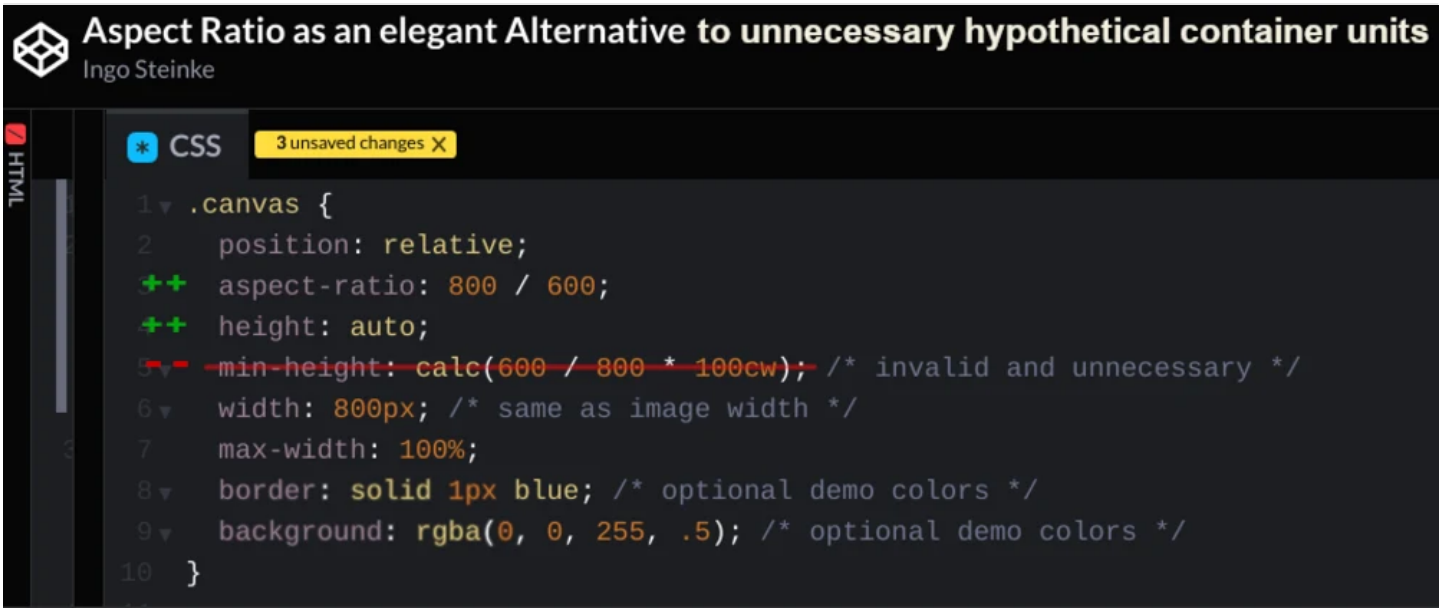


Ingo Steinke
Posted on 29 Dec 2021 • Updated on 3 Jan

Aspect ratio: no need for container units!

#css #html #webdev #todayilearned

What's next in CSS? (3 Part Series)

1. CSS :has(.parent-selectors)
2. **Aspect ratio: no need for container units!**
3. Animated Gradient Text Color

When I first heard about container queries, I was mistaken that there would be **"container units"** in CSS, probably a unit like percent, but relative to the other axis. "100cw" would always be 100% of the container width, even if used insight a `height` property.

Trying to make Sense of a Misconception 🤔🤓🤣

I tried to make sense of the hype. Container units could possibly a handy feature. I even had an actual use case in a customer project this year.

Disclaimer: there will be no container units in CSS in the near future (and probably never), because we probably don't need them. At least there have been alternatives for every of my use cases so far.

Let's have a look at my recent use case:

A Carousel Slideshow inside a Column 🎠🎪

This is an actual requirement I had in 2021. While I would rarely recommend using slideshow carousels at all, this is just my personal opinion as a front-end developer. I am no usability expert, and some of my customer (or rather the teams coming up with the designs) seem to love anything that moves on a website.

Let's just imagine we want to style a slideshow.

Dynamic Height Calculation relative to Container Width

The slideshow container is a block element that needs to keep its proportions without stretching or clipping its content, by setting the height in proportion to the width. Just like HTML image elements behave in the regular document flow.

```
<img width="800" height="600" alt="an image" src="img.jpg">

<style>
img {
  max-width: 100%;
  height: auto;
}
</style>
```

Unless they are taken out of the flow by using `position: absolute` or a CSS transformation, which is a typical side effect of a slideshow / carousel. Have a look at this simplified code example:

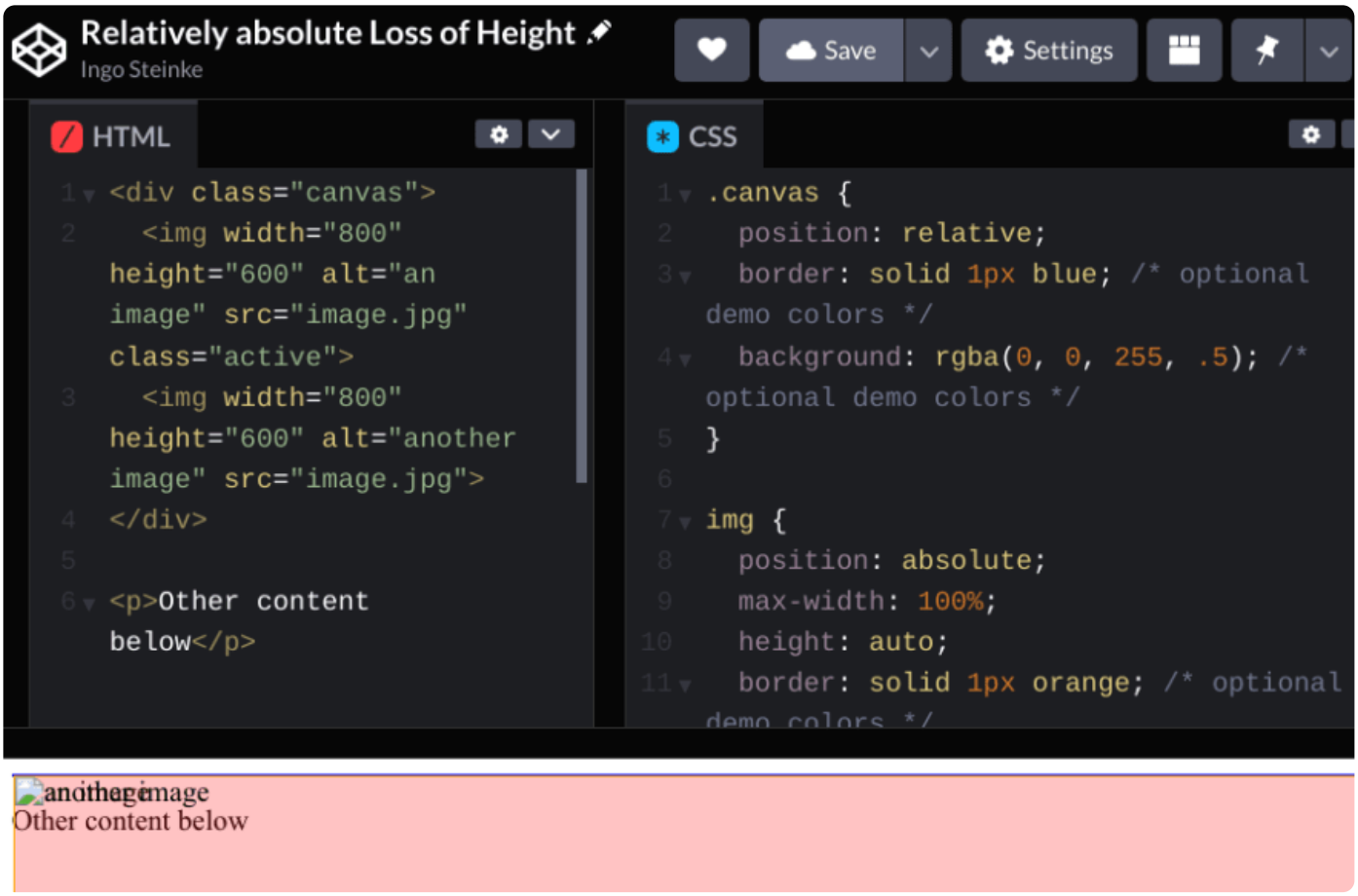
```
<div class="canvas">
  <img width="800" height="600" alt src="img.jpg" class="active">
  <img width="800" height="600" alt src="img2.jpg">
</div>

<style>
```

```
.canvas {
  position: relative;

}

img {
  position: absolute;
  max-width: 100%;
  height: auto;
}

image:not(.active) {
  display: none;
}

</style>
```

This is a very basic abstraction of a slideshow markup. A series of images, usually only one of them (the "active" image) visible at a time, while the others will be revealed by scrolling, dragging, swiping or waiting for an automated animation.

But this example is enough to reproduce a typical challenge that web developers face when dealing with absolutely positioned content.

Relatively absolute Loss of Height



This codepen shows how our image container has lost its height instead of growing to the height of the images inside.

Out of the flow, our images now behave like butterflies fyling over the content which they used to be a part of.

🐛 ➤➤➤ 🦋✨ ➤➤➤ 🦄🎪

This is good for the images (because we are free to move and position them anywhere, which is necessary for the desired slideshow effect), but it' bad for our document. The content below our image container now overlaps our slideshow, while it should stay visually below like it did before.

🚧🚧🚧🚧🚧

Keeping the Distance

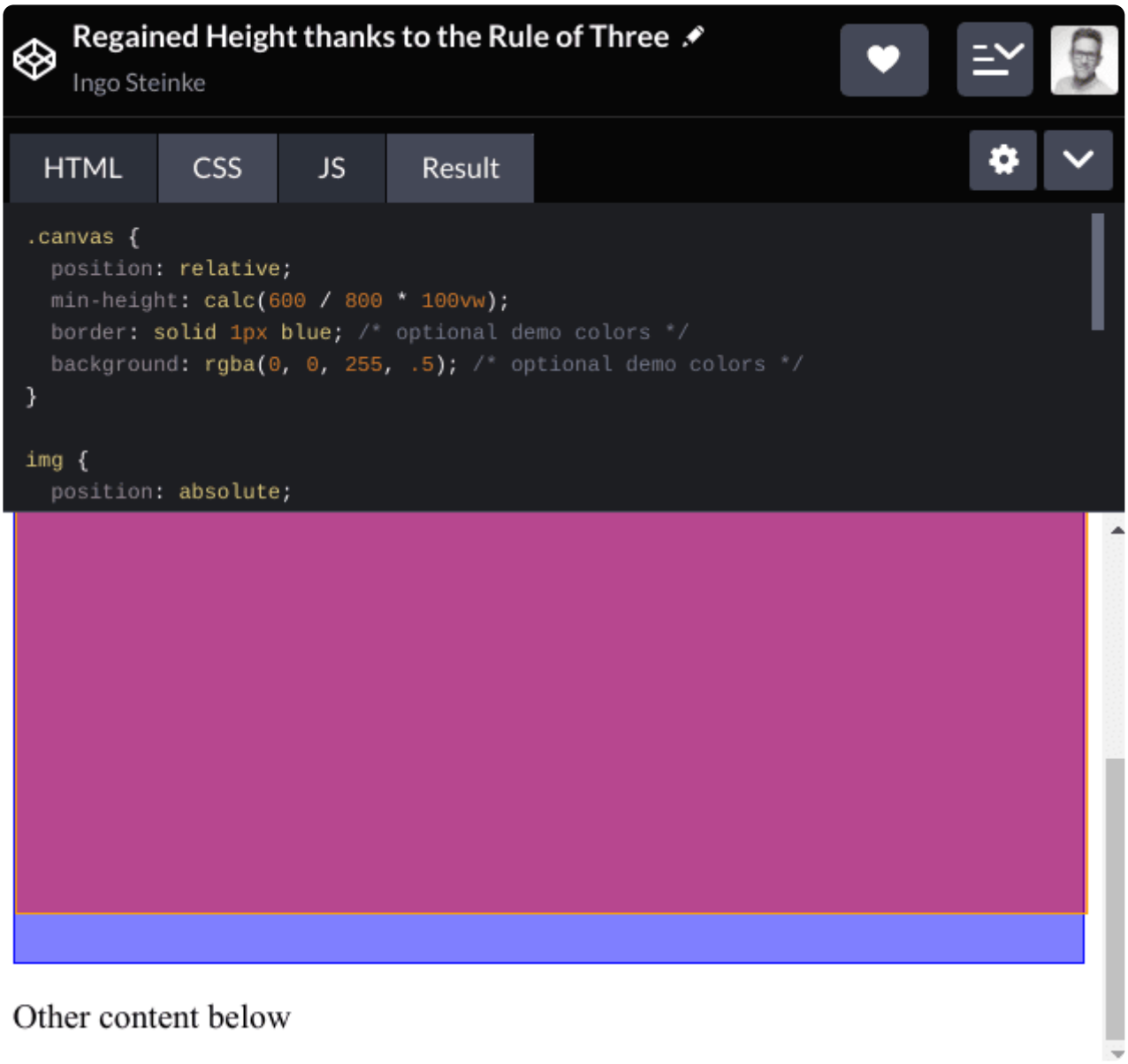
We can set a minimal height, but as the images are dynamically sized (they will shrink when the window is resized to prevent being cut off on small screens like on a mobile device), we can't use a static pixel value here.

Adapting to the Viewport Width

As long as we only consider the viewport width, we could use viewport units inside a calculation. "Viewport" is the part of the browser that displays our current document. `100vw` means 100% of the available viewport width. This is a simple rule of three calculation:

```
.canvas {
  min-height: calc(800 / 600 * 100vw);
```

Did this fix our problem?



Now we have a height that behaves mostly as expected. We should to fix the additional space below the image, probably due to some unintended inline display behavior or whatever. But the basic concept does its job.

Unless we want to adapt to a parent container.

Responsive Design inside Container Elements

Maybe our slideshow will be placed inside a column next to a navigation or some ad banners, so we must not rely on the `100vw` viewport width.

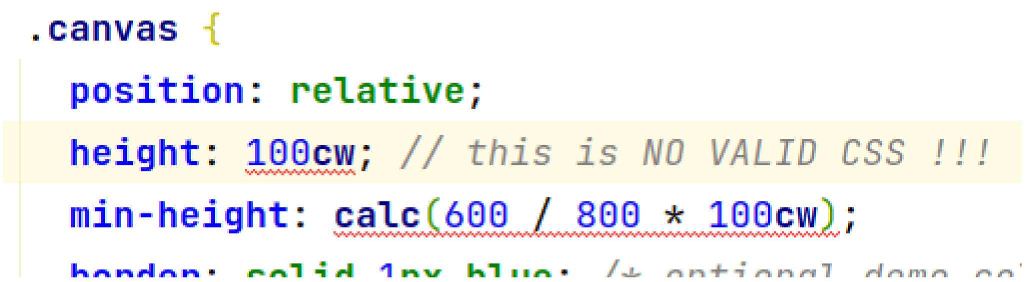
Let's just use percentage then, right? No!

While `100vw` is always 100% of the viewport width (don't care about the scrollbar details either), `100%` is always 100% or the current axis, so

`width: 100vw` and `width: 100%` are the same on a top level, but `height: 100vw` vs. `height: 100%` are totally different, as these 100% will be 100% of the height, not of the width!

Container Units? 📊📐

One might conclude, that we need container units, similar to the viewport units, for our use case, like:



This is no valid CSS (and will probably never be):

```
.canvas {
  position: relative;
  height: 100cw; // this is NO VALID CSS !!!
  min-height: calc(600 / 800 * 100cw);
```

I just made up this pseudo-code. Don't copy!
There is a better alternative!

Aspect Ratio as an elegant Alternative

A more elegant alternative to [set a div height from its width]() is the `aspect-ratio` property.

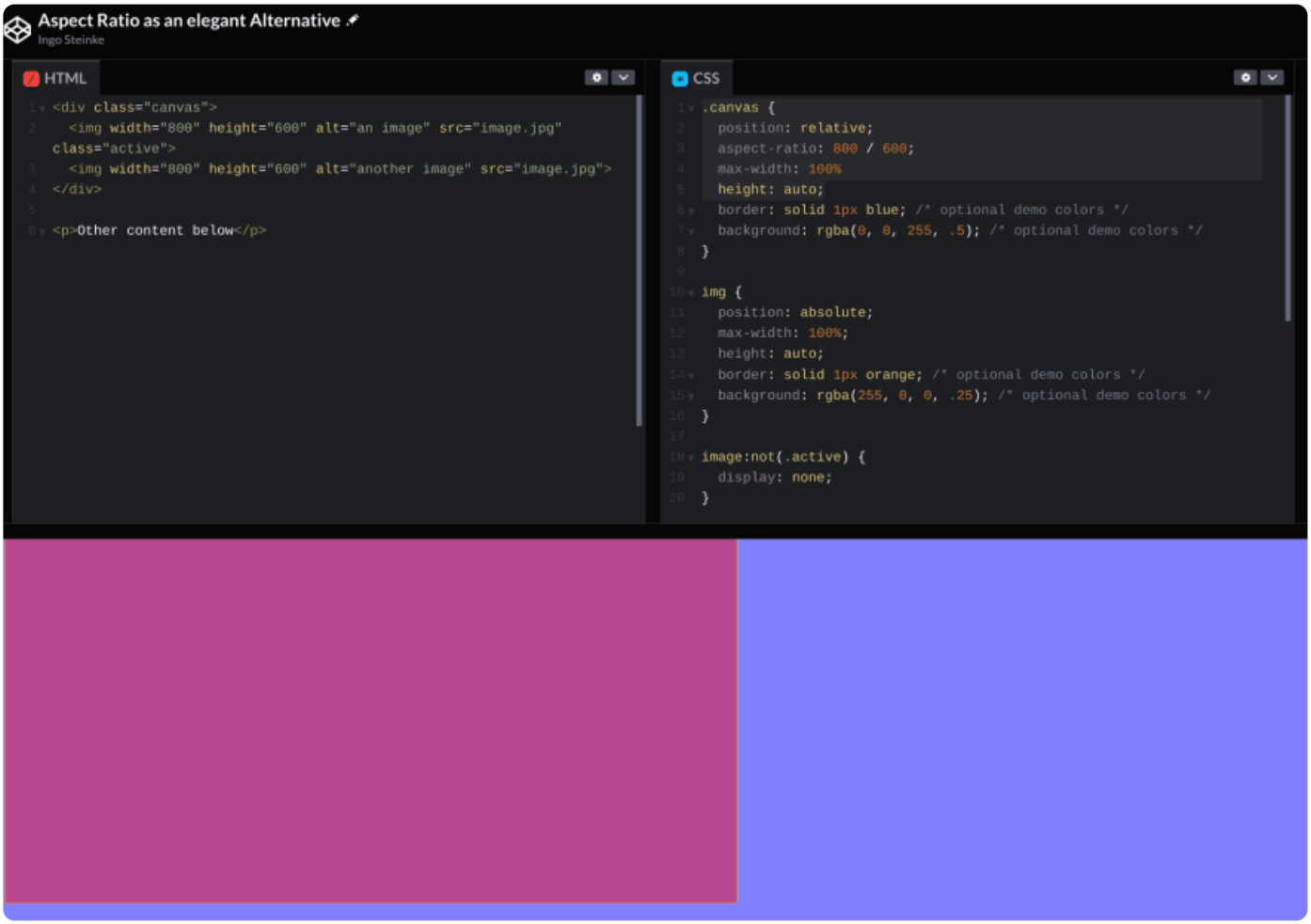
Why is this elegant? We don't need to define calculations. Instead, we write compact code, that is easy to use and understand. Let's leave it to the browsers to do the proper math, just like they have been handling images properly for decades.

[Aspect ratio]() makes any block element behave just like an image, so we can use it to adapt its width to the viewport and keep its ratio to figure out the appropriate height:

```
.canvas {
  position: relative;
  aspect-ratio: 800 / 600;
  max-width: 100%
  height: auto;
```

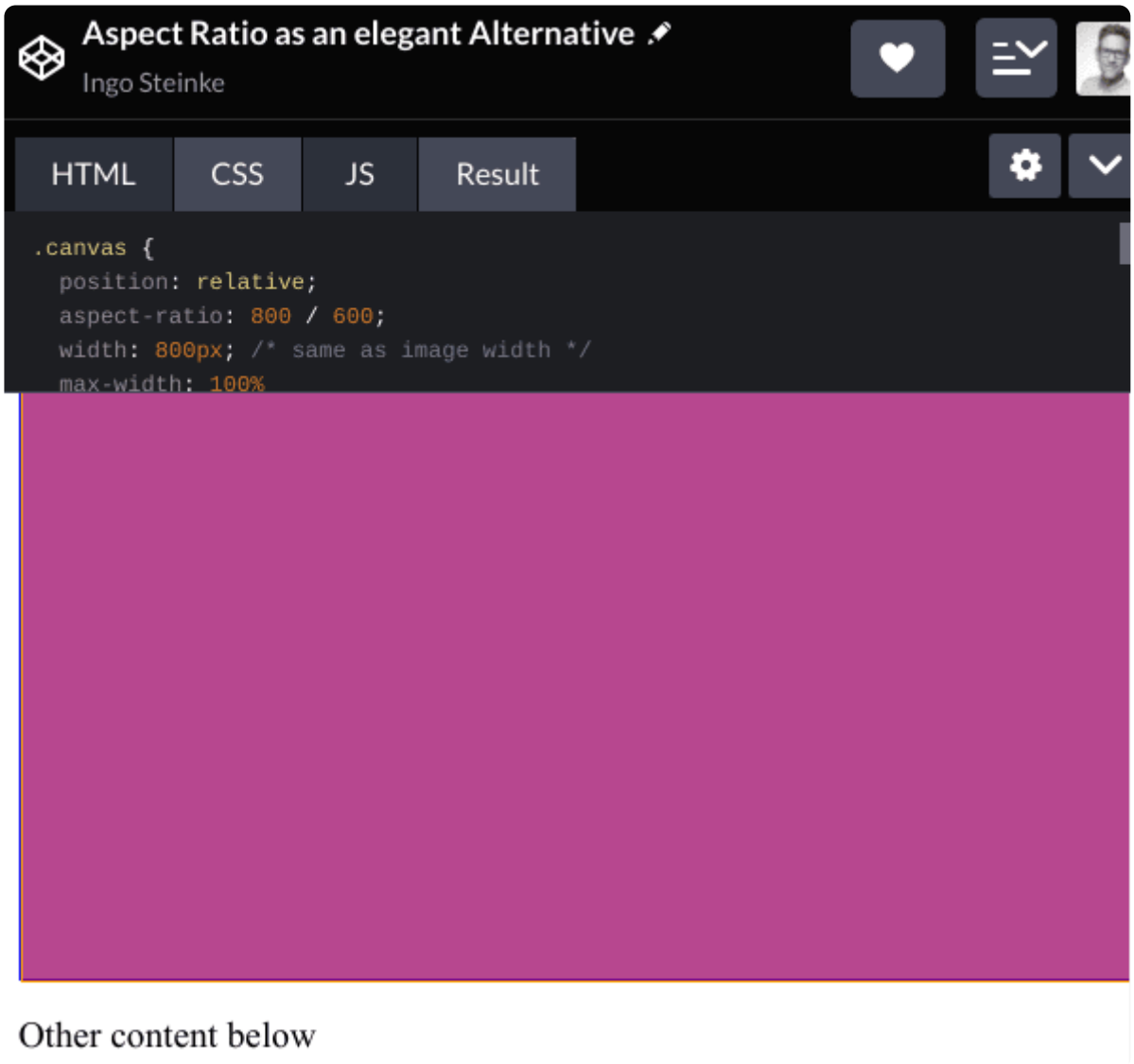
we still have one problem here: max width should be restricted both to the viewport or parent container width and to the image width.

The code example above will make our `.canvas` div too large when the container is wider than the image width:

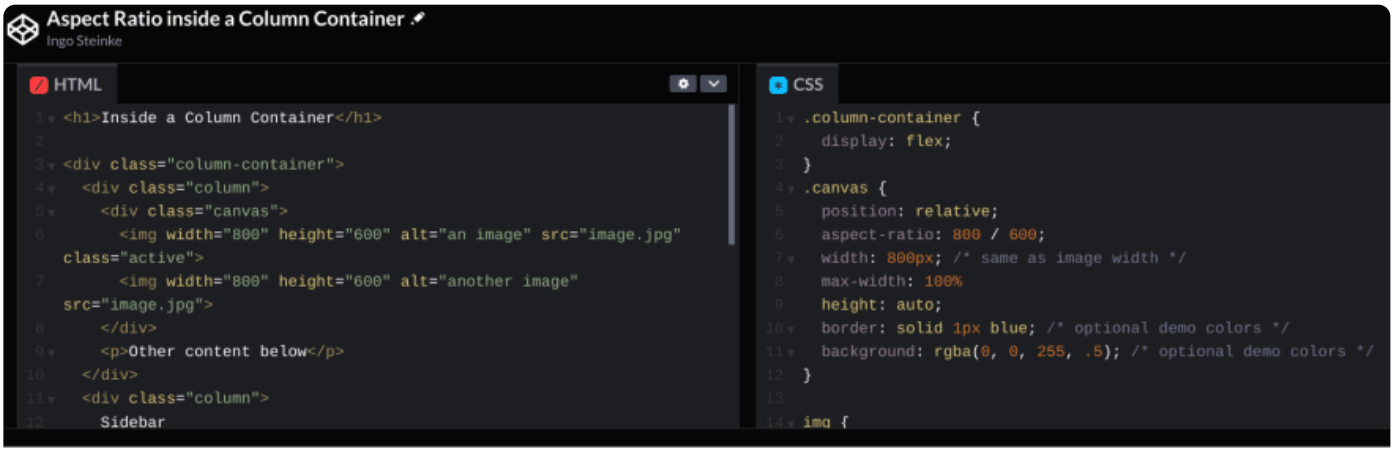


We have to set our container `width` to the same width as that of our image element to make it work.

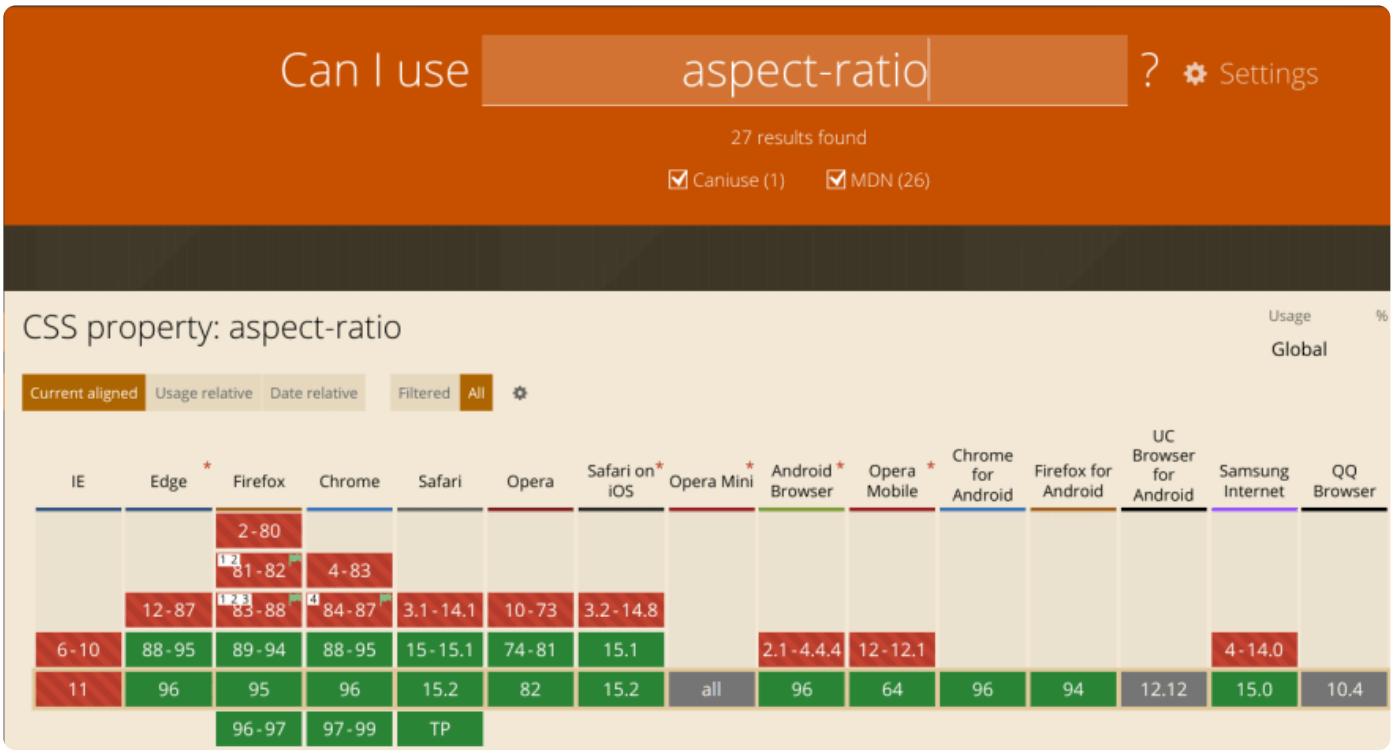
This will resize properly...



... and it should also work inside of a column, when the parent container width is not the full viewport width. That's the use case I described at the beginning of this article.



Can I use aspect-ratio?

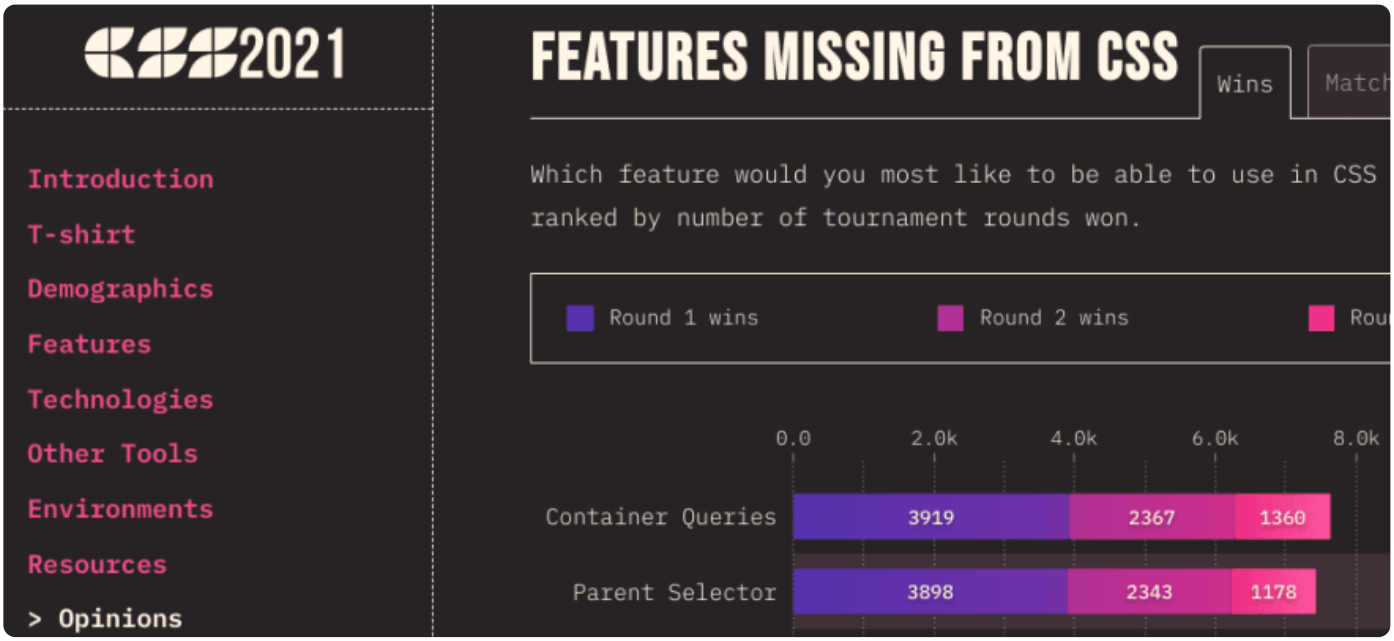


The CSS property `aspect-ratio` has fairly good browser support. You could still use the min-height calculation as a fallback for unsupported browsers.

Conclusion

There is no need for container units.

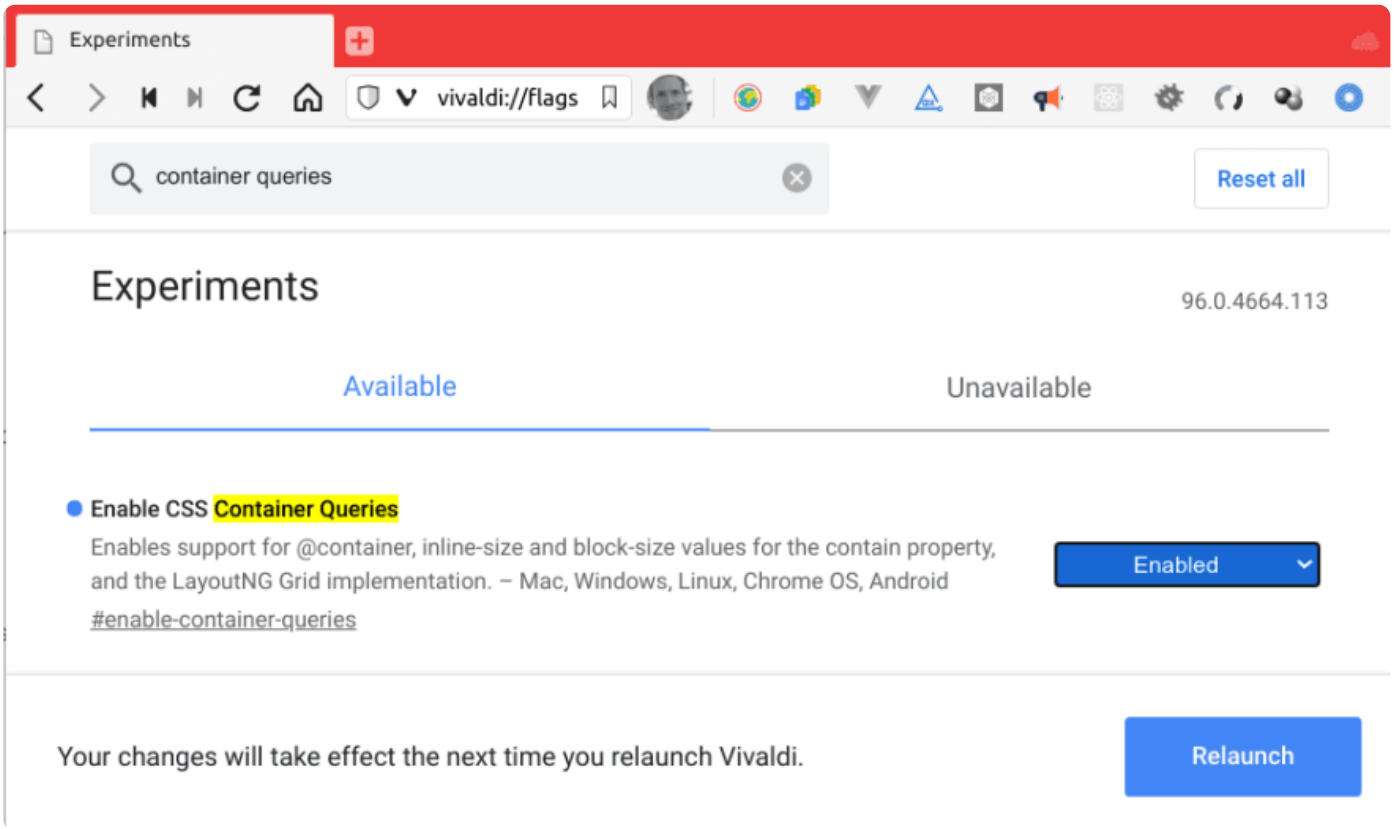
What we do need for better responsive frontend web development are **container queries**, one the most popular missing CSS language feature [according to the state of CSS survey 2021](#).



This cornucopia of container queries by CSS tricks is a good way to understand the upcoming update which can already be tested after setting a feature flag in all Chromium-based browsers like Google Chrome, Microsoft Edge, Opera, and Vivaldi.

Enable CSS Container Queries

Type `chrome://flags` in the address bar, and switch "Enable CSS Container Queries" to "enabled".



Restart your browser and you can start to experiment with container queries.

```
/* stylelint-disable-next-line scss/at-rule-no-unknown */
@container (max-width: 50vw) {
  flex-wrap: wrap;
}
```

I will follow-up on container queries experiments in this blog series.

What's next in CSS? (3 Part Series)

1. CSS :has(.parent-selectors)
2. **Aspect ratio: no need for container units!**

3 Animated Gradient Text Color

Discussion (0)

Code of Conduct • Report abuse

Ingo Steinke

Web Development, Sustainability, Art and Music, Nature and Travel, Sustainability

LOCATION
Germany

WORK
Creative Web Developer at Ingo Steinke

JOINED
21 Sep 2019

More from Ingo Steinke

Animated Gradient Text Color
#webdev #showdev #css #tutorial

CSS :has(.parent-selectors)
#css #webdev #todayilearned #javascript

Printable Lazy Loading
#webdev #html #performance #a11y