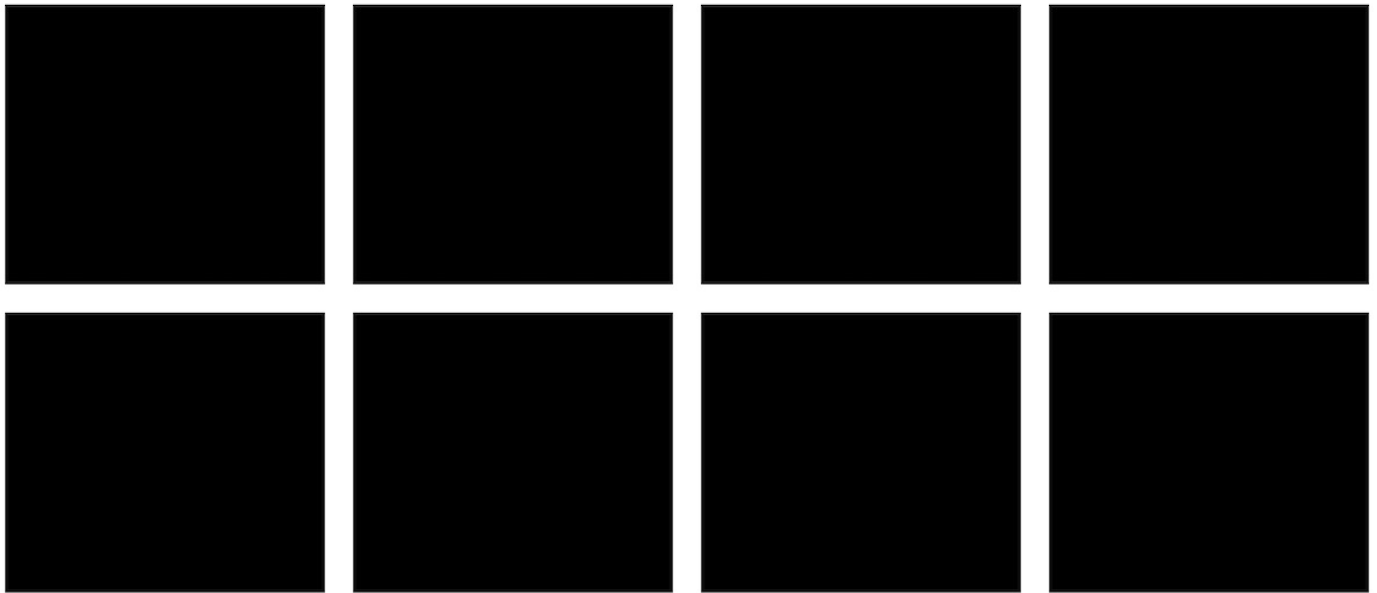


Intrinsically Responsive CSS Grid with `minmax()` and `min()` (<https://evanminto.com/blog/intrinsically-responsive-css-grid-minmax-min/>)

Published July 11, 2019



[CSS Grid](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout) (https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout) is now widely supported across modern browsers, and there are lots of folks doing great work with it! But unfortunately, one of the most useful features of the specification doesn't quite work as advertised. Specifically, it's not possible to create an "intrinsically responsive grid" — that is, **a grid that is responsive based on the size of its container, without the use of media queries**. But thanks to some standards that are now available in some browsers and on their way to others, we can fix that!

Before I explain the solution, let's review the problem. With CSS Grid, we can create a grid of items arranged in equally sized columns and tell those column sizes to adjust based on the amount of

available space:

HTML

CSS

Result


EDIT ON

```
ul {
  display: grid;
  grid-gap: 1rem;
  grid-template-columns: repeat(auto-fill,
minmax(10rem, 1fr));

  background: lightgray;
  list-style: none;
  padding: 0;
}

ul > * {
  background: black;
  height: 10rem;
}

.narrow {
  margin-left: auto;
}
```

Run Pen 

Resources 1x 0.5x 0.25x Rerun

This technique uses a number of features in the Grid specification:

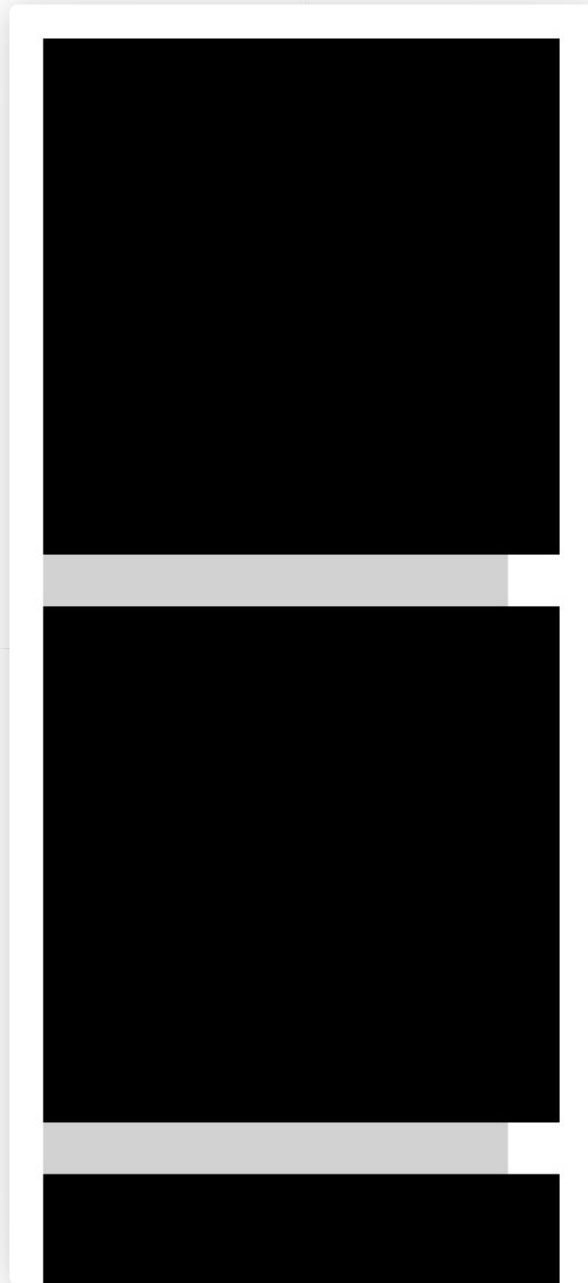
`repeat()` tells Grid to create multiple tracks with the same size parameters (not necessarily the same size, though in this case they all end up being the same).

`auto-fill` tells it to create as many tracks as necessary to fill the space.

`minmax(10rem, 1fr)` tells it to determine each track's size by finding a value between a minimum of 10rem and a maximum of 1fr. The `fr` unit is what makes sure that the tracks are allowed to grow to fill any remaining space.

Thanks to all this, the container adds new columns as it grows. A key feature of this solution is that the columns change based on the *container* size, not the viewport size. That means we can apply this to a reusable component and be confident that it will always look right, without the need for media queries overriding the default behavior on specific pages. So what's the problem?

Responsive... Sorta



Since each grid track has a *minimum* size of `10rem`, they can't shrink below that size. That means when the container is smaller than `10rem`, the grid items overflow the container! To fix this, we'd have to wrap the `grid-template-columns` declaration in a media query, like so:

HTML

CSS

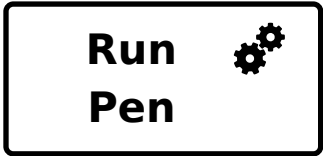
Result

EDIT ON

```
ul {  
  display: grid;  
  grid-gap: 1rem;  
  grid-template-columns: 1fr;  
  
  background: lightgray;  
  list-style: none;  
  padding: 0;  
}  
  
@media (min-width: 12em) {  
  ul {  
    grid-template-columns: repeat(auto-  
fill, minmax(10rem, 1fr));  
  }  
}  
  
ul > * {
```

Run

Pen



Resources1x0.5x0.25xRerun

But now that we're in media query-land, our grid only switches when the *viewport* is small. We can't put it inside a small *container*, or we run into the overflow problem all over again. This is why [Heydon Pickering](https://twitter.com/heydonworks) (<https://twitter.com/heydonworks>) and [Andy Bell](https://twitter.com/andybelldesign) (<https://twitter.com/andybelldesign>)'s [Every Layout](https://every-layout.dev) (<https://every-layout.dev>) site generally avoids media queries and CSS Grid, instead achieving intrinsic responsiveness through Flexbox.

Thankfully, with some new additions to the CSS spec that have started landing in browsers, we can fix this issue, allowing CSS Grid to live up to its full potential as a tool for [intrinsic web design](http://www.zeldman.com/2018/05/02/transcript-intrinsic-web-design-with-jen-simmons-the-big-web-show/) (<http://www.zeldman.com/2018/05/02/transcript-intrinsic-web-design-with-jen-simmons-the-big-web-show/>).

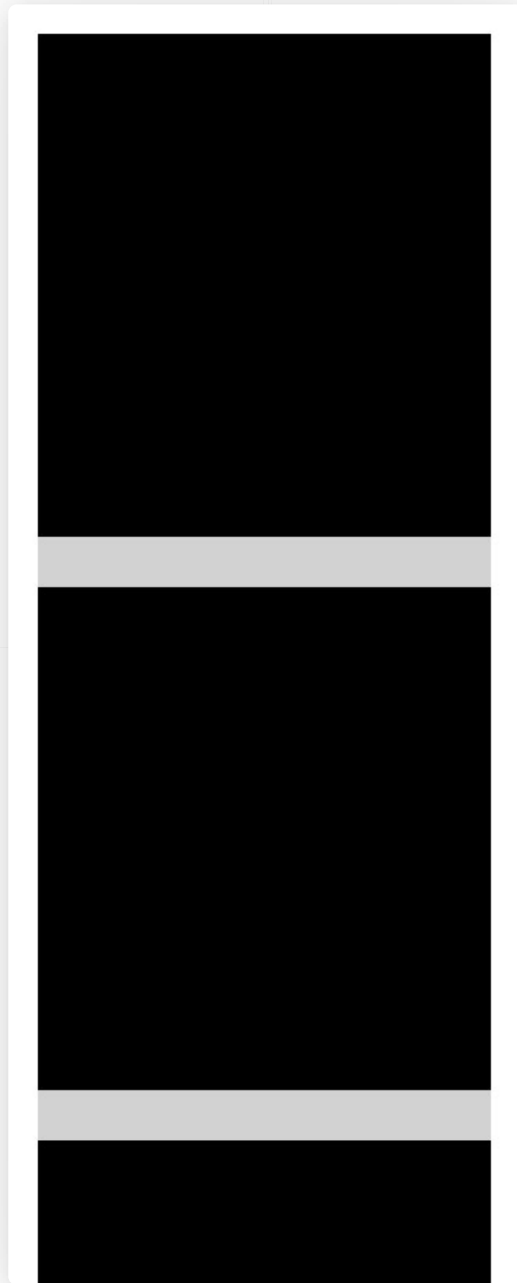
The Intrinsically Responsive Grid

The previous example ran into problems because we were setting a fixed value as the minimum track size. When the container size shrunk, the items didn't shrink with it, and thus: overflow. But what if we could set a minimum size that would shrink based on the container size, ensuring that our tracks were never too big for their container?

As luck would have it, we can do just that, with the help of the `min()` function (<https://developer.mozilla.org/en-US/docs/Web/CSS/min>):

```
grid-template-columns: repeat(auto-fill, minmax(min(10rem, 100%), 1fr));
```

`min()` accepts one or more values and returns the smallest value. The magic of the function is that, just like `calc()`, the arguments can use different units, which allows us to return values that change dynamically based on context. In this case, we're returning the current width of the container, capped at a maximum value of `10rem`.



`min()` is one of three new comparison functions introduced as part of the [CSS Values and Units Module Level 4](https://drafts.csswg.org/css-values-4/#comp-func) (<https://drafts.csswg.org/css-values-4/#comp-func>). There's also `max()`, which naturally does the inverse of `min()`. Finally `clamp()` is a convenience function that applies both a minimum *and* a maximum to a single value.

In this example, there are two cases we have to worry about, and `min()` solves for both of them. When the container is *smaller* than `10rem`, the minimum value in our `minmax()` expression is set to the full width of the container, giving us a nice single-column view with no overflow. When the container is *larger* than `10rem`, the minimum value is set to `10rem`, so we get a responsive grid with equal tracks of at least `10rem` each.

In the Real World

I know what you're thinking. "This sounds cool and all, but I can't actually *use* it yet." You're only half right.

`min()` isn't very well supported, but it *is* available in at least one browser: Safari quietly implemented both `min()` and `max()` a few versions back! Unfortunately there's no `clamp()` yet, but you can simulate it using the other two. Plus, Tab Atkins recently announced that [Chrome support is on its way](https://twitter.com/tabatkins/status/1141830941628780544) (<https://twitter.com/tabatkins/status/1141830941628780544>). If you want to keep track of browser support, [CanIUse has a page for all three functions](https://caniuse.com/#feat=css-math-functions) (<https://caniuse.com/#feat=css-math-functions>).

With that good news out of the way, here's a working version of the code above (open in Safari to see it in action):

HTML

CSS

Result


EDIT ON

```
<h2>Full Container</h2>
<ul>
  <li></li>
  <li></li>
  <li></li>
  <li></li>
  <li></li>
  <li></li>
  <li></li>
  <li></li>
</ul>

<h2>Narrow Container</h2>
<ul class="narrow">
  <li></li>
  <li></li>
  <li></li>
  <li></li>
```

Run

Pen



Resources

1x 0.5x 0.25x

Rerun

One browser isn't great, but it doesn't mean you have to give up on this solution. For now, you can layer it onto your existing code as a progressive enhancement. Here's my preferred solution:

HTML

CSS

Result

EDIT ON

```
ul {
  display: grid;
  grid-gap: 1rem;

  /* Partially responsive fallback */
  grid-template-columns: repeat(auto-fill,
minmax(calc(10% + 7.5rem), 1fr));


  /* Fully responsive version */
  grid-template-columns: repeat(auto-fill,
minmax(min(10rem, 100%), 1fr));

  background: lightgray;
  list-style: none;
  padding: 0;
}

ul > * {
```

Run

Pen



Resources1x0.5x0.25xRerun

This one uses a `calc()` expression as a fallback. The key part is the `calc(10% + 7.5rem)`, which combines a fixed minimum with a percentage width. By computing the minimum based on *both* of these, we lower the threshold for triggering overflow. Now it will trigger when the container is slightly larger than `7.5rem`, whereas in our earlier solution it would trigger at `10rem`. This solution is a bit finicky though, so you'll want to adjust both the percentage and fixed values to fit your grid's design.

With this as our fallback, we can take advantage of CSS's ability to override duplicate property declarations by re-declaring `grid-template-columns`. Browsers that support `min()`, namely Safari, will get the fully responsive solution, while Chrome, Firefox, Edge, and others will get the fallback. As more browsers add support for `min()`, they will automatically switch over to the new layout.

Wrapping Up

The `min()`, `max()`, and `clamp()` functions are really powerful tools, allowing us to toggle between values in real time based on changing conditions. Being able to test them out in a real browser opens up lots of opportunities for exploration, and I'm confident that this little fix for CSS Grid is just