

CSS GRID LAYOUT

FINALLY!

Manuel
Matuzović
@mmatuzo 

Overview

- What is Grid and why do we need it.
- Browser support: Can we use it today?
- New properties, units and keywords.
- CSS Box Alignment Module.
- CSS Intrinsic And Extrinsic Sizing.
- Accessibility.

Overview

- Common patterns.
- Stacking order.
- Responsive Webdesign.
- Progressive Enhancement.
- Debugging.
- The future of Grid (Level 2).

WHAT'S CSS GRID LAYOUT?

What's CSS Grid Layout?

- The first true layout method in CSS.
- `float`, `display: inline-block`, `position`, `display: table` **were originally not intended for building layouts.**
- Two-dimensional layouting.
- Grid structures are described in CSS and not in HTML.
- Kinda like table layouts, but in CSS, responsive and flexible.

History and background

- Early 90s: First ideas, but “too complex” to implement.
- 1996 “frame-based” layout model.
- 2005 Advanced Layout Module, which later turned into the Template Layout Module.
- Later Microsoft was in the planning stages of Windows 8 and were going to enable building apps with web technologies.

History and background

- Microsoft shipped a grid layout implementation behind the `-ms-` vendor prefix in Internet Explorer 10 in 2011.
- They presented a draft Grid Layout spec to the W3C in 2012.
- The spec that shipped is not the same spec we have today. Among other differences, it has no lines, auto placement or templates.
- CSS Working Group began tweaking Microsoft's proposal. The plan was to unite Microsoft's proposal, Bos' ideas in the Advanced Layout Module and the idea of *drawing lines*.

History and background

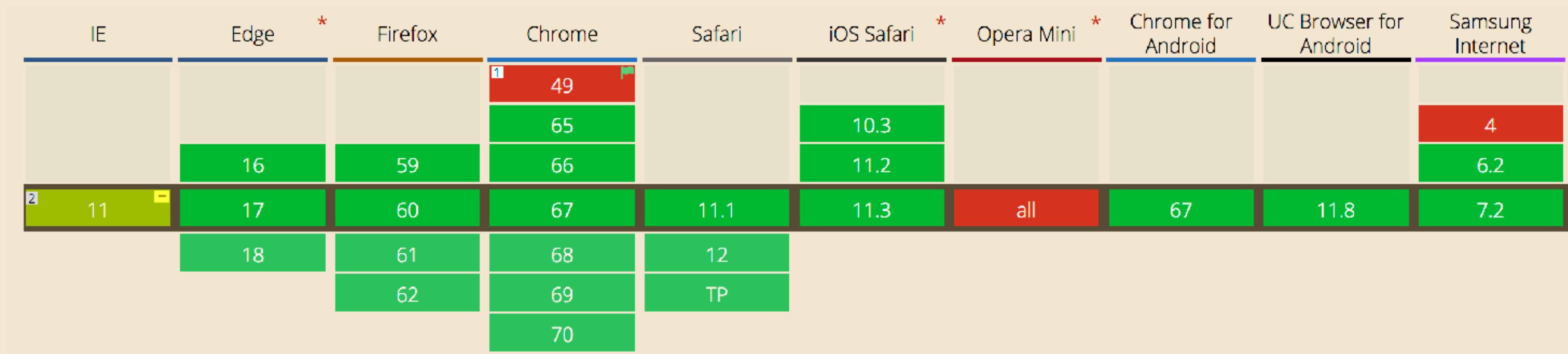
- The media company Bloomberg hired Igalia, an open source consultancy, to implement CSS Grid for both Blink and WebKit.
- Even with two ready-made implementations, some browsers were still concerned the feature wouldn't find its footing.
- But, a good number of designers and developers were starting to get excited about Grid and began to put pressure on browser vendors to implement it.
- Especially demos and articles by Rachel Andrew and Jen Simmons got people really exited.

History and background

- Google landed an implementation of CSS Grid in Chromium 56 for Android in January of 2017.
- Chrome and Firefox implementations in early March.
- Opera and Safari by the end of March 2017.
- Ironically, the last company to ship CSS Grid was Microsoft (October 17th).
- All major browsers rolled out their implementations in a single year!

BROWSERSUPPORT?

Browsersupport



Browsers support

- All major desktop and mobile browsers.
- IE 10 - Edge 15: Old Grid specification.
- Edge 16+ both specs.

**IS GRID A REPLACEMENT
FOR FLEXBOX?**

Is Grid a replacement for Flexbox?

Flexbox

With Flebox **content dictates the layout**. For example, we use it for even distribution of items or filling available space with items.



Is Grid a replacement for Flexbox?

Grid

With grid **the layout has to be defined**. We define columns and rows.
Items are explicitly or implicitly placed in those rows.

Is Grid a replacement for Flexbox?

Flexbox

...for one dimension (column or row).

...when child items dictate the layout.

...if you want to distribute the available space.

Is Grid a replacement for Flexbox?

Grid

...for two dimensions (column and row).

...when the parent item dictates the layout.

...as soon as flex items are losing their flexibility.

*„If you are adding widths to all your flex items,
you probably need grid“*

– Rachel Andrew | Render 2017

GRID BASICS

grid, tracks, cells, gaps

Grid Basics

LEGEND

The element where **display** has been set to **grid**.

The grid container.

A direct child item of **.container**.

```
.container {  
  ...  
}
```

```
.grid-item {  
  ...  
}
```

A screenshot of a web browser window showing a CodePen editor for a CSS grid layout. The page title is "CSS Grid Layout Basics: columns, rows, gaps". The main content area displays a red box containing nine elements labeled "Element 1" through "Element 9". The sidebar on the right contains four panels: "HTML", "CSS", "JS", and "Console".

HTML

```
<div class="grid">
  <article class="item">
    <h2>Element 1</h2>
  </article>
  <article class="item">
    <h2>Element 2</h2>
  </article>
  <article class="item">
    <h2>Element 3</h2>
  </article>
  <article class="item">
    <h2>Element 4</h2>
  </article>
  <article class="item">
    <h2>Element 5</h2>
  </article>
  <article class="item">
    <h2>Element 6</h2>
  </article>
  <article class="item">
    <h2>Element 7</h2>
  </article>
  <article class="item">
    <h2>Element 8</h2>
  </article>
  <article class="item">
    <h2>Element 9</h2>
  </article>
</div>
```

CSS

```
.grid {
```

JS

At the bottom of the sidebar, there are buttons for "Console", "Assets", "Comments", and a search bar. On the far right, there are buttons for "Delete", "Share", "Export", "Embed", and "Collections".

COLUMNS, ROWS & GAPS

bit.ly/grid_basic1

Grid Basics

Defining grids.

Values:

grid

inline-grid

```
.container {  
    display: grid;  
    display: inline-grid;  
}
```

/ Starting with level 2 of the spec */*

```
.container {  
    display: subgrid;  
}
```

Grid Basics

Adding rows
(explicit grid)

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

Adding columns
(explicit grid)

```
.container {  
    grid-template-columns: 200px 200px 150px;  
}
```

Grid Basics

Gaps (gutter)

```
.container {  
    grid-gap: 20px;  
}
```

Gaps for each direction

```
.container {  
    grid-column-gap: 10px;  
    grid-row-gap: 10px;  
}
```

The screenshot shows a web browser window displaying a CodePen project titled "CSS Grid Layout Basics: flow". The project page has a dark header with the title and a "PRO" badge. Below the header, there's a navigation bar with links like "Save", "Fork", "Settings", and "Change View". The main content area features a 3x3 grid of red rectangular boxes, each containing white text. The first row contains "Element 1", "Element 4", and "Element 7". The second row contains "Element 2", "Element 5", and "Element 8". The third row contains "Element 3", "Element 6", and "Element 9". To the right of the grid, there are three code editors: "HTML", "CSS", and "JS". The "HTML" editor shows the structure of the page with three nested article elements. The "CSS" editor contains the following code:

```
.grid {  
  display: grid;  
  grid-template-columns: 200px 200px 200px;  
  grid-template-rows: 200px 200px 200px;  
  grid-auto-flow: column;  
  grid-gap: 20px;  
}
```

The "JS" editor is currently empty.

FLOW

bit.ly/grid_flow

Grid Basics

Controlling
auto-placement

Values:

row (default)

column

dense

```
.grid {  
  grid-auto-flow: column;  
}
```

A screenshot of a CodePen editor interface. The title bar says "Limited number of items per column". The main content area displays a grid of 11 items arranged in three rows. The first row has four items: "item 01", "item 04", "item 07", and "item 10". The second row has three items: "item 02", "item 05", and "item 08". The third row has four items: "item 03", "item 06", "item 09", and "item 11". Each item is a red button-like element containing its respective text. To the right of the preview, there are tabs for "HTML", "CSS", and "JS". The "HTML" tab shows the following code:

```
<ul class="grid">
  <li class="grid-item"><a href="#">item 01</a></li>
  <li class="grid-item"><a href="#">item 02</a></li>
  <li class="grid-item"><a href="#">item 03</a></li>
  <li class="grid-item"><a href="#">item 04</a></li>
  <li class="grid-item"><a href="#">item 05</a></li>
  <li class="grid-item"><a href="#">item 06</a></li>
  <li class="grid-item"><a href="#">item 07</a></li>
  <li class="grid-item"><a href="#">item 08</a></li>
  <li class="grid-item"><a href="#">item 09</a></li>
  <li class="grid-item"><a href="#">item 10</a></li>
  <li class="grid-item"><a href="#">item 11</a></li>
</ul>
```

The "CSS" tab shows the following code:

```
.grid {
  display: grid;
  grid-template-rows: auto auto auto;
  grid-auto-flow: column;
  grid-gap: 14px;
}
```

LIMITED NUMBER OF ITEMS PER COLUMN

bit.ly/grid_flow2

Grid Basics

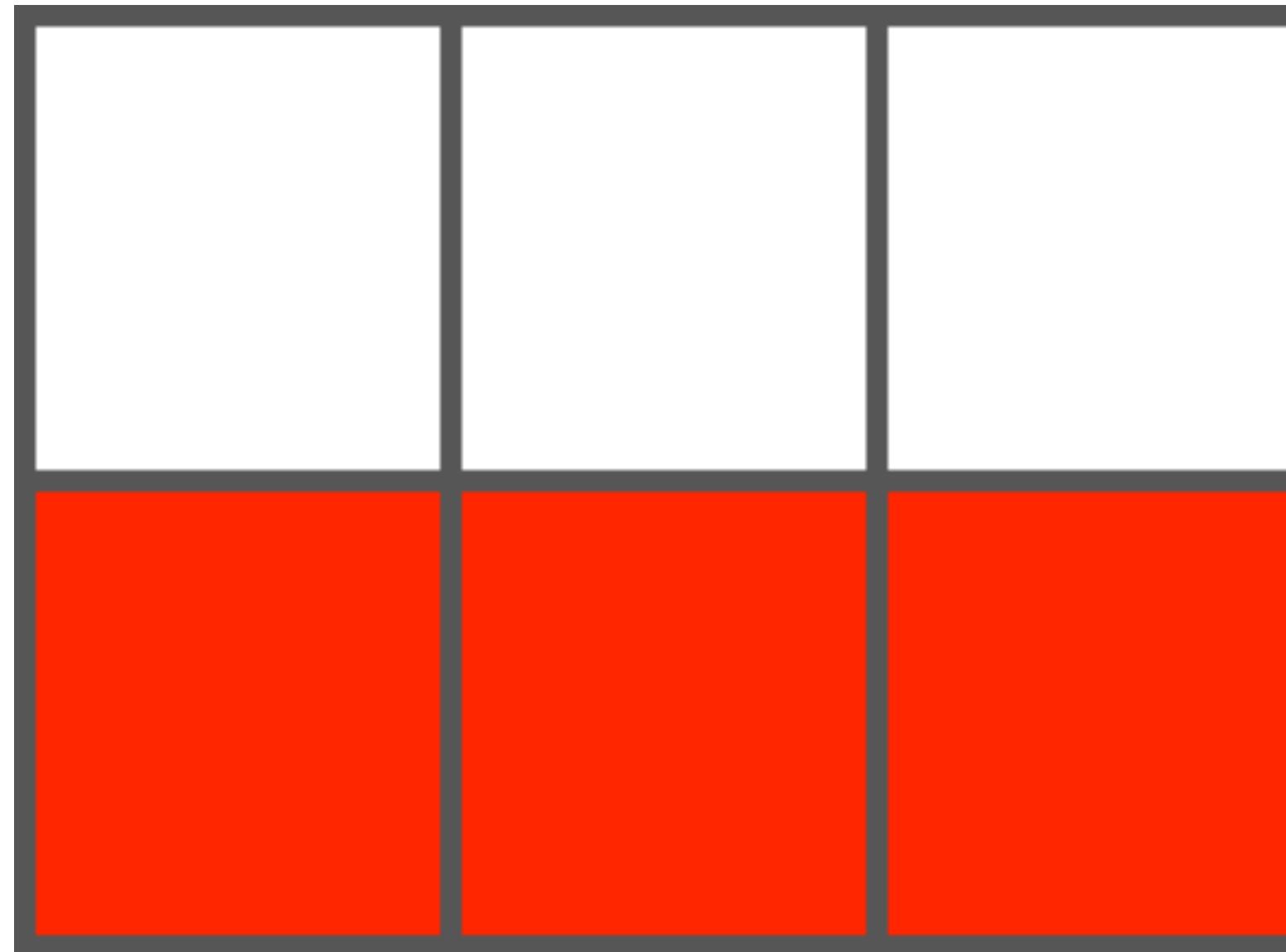
A “dense” packing algorithm, which attempts to fill in holes earlier in the grid if smaller items come up later.

May cause a disconnect between DOM and visual order. Use it deliberately.

```
.grid {  
  grid-auto-flow: dense;  
}
```

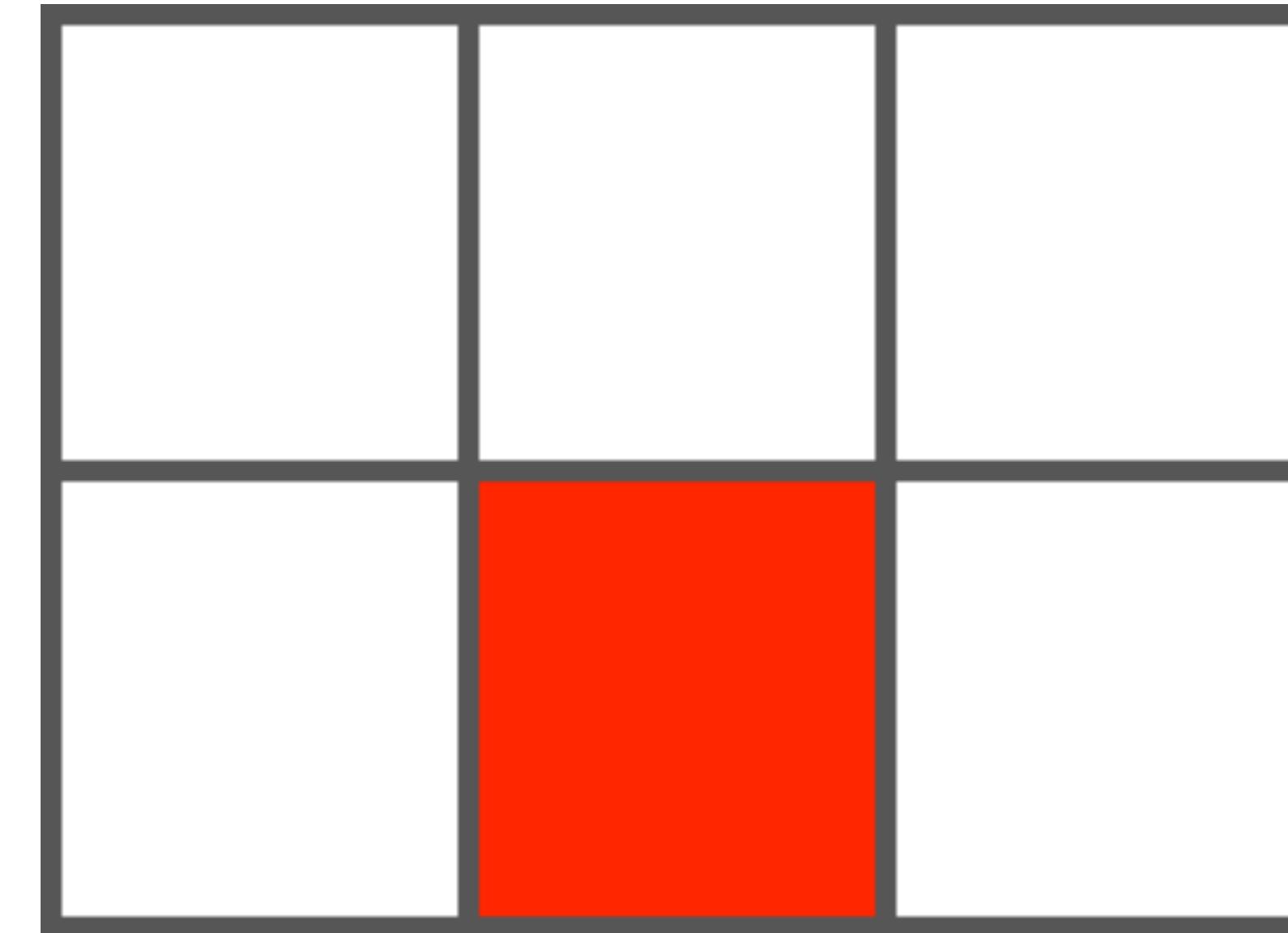
Terminology

TRACK



Either horizontal or vertical

CELL



A single cell within a track

Illustrations: <https://gridbyexample.com/what>

The screenshot shows a web browser window with three tabs: "Implicit grid", "CSS Grid Layout Basics: column", and "CodePen - CSS Grid Layout - N". The main content area displays a grid layout with numbered boxes (1-12) and text definitions for grid terms.

Grid lines are the horizontal and vertical dividing lines of the grid.

CSS Grid Layout

New Terminology

GRID AREA
A grid area consists of one or more adjacent grid cells. It is bound by four grid lines, one on each side of the grid area.

GRID TRACK (COLUMN)
A grid track is the space between two adjacent grid lines, forming a grid column or grid row.

GRID CELL
A grid cell is the smallest unit of the grid. A grid cell is the intersection of a grid row and a grid column.

GRID TRACK (ROW)
A grid track is the space between two adjacent grid lines, forming a grid column or grid row.

As you can see, grid tracks can overlap

CSS Grid Layout works in Firefox 52+, Safari 10.1 (and iOS), Chrome 57+ (and Chrome 57 for Android), Opera and Edge.

EXPLICIT GRID VS. IMPLICIT GRID

A screenshot of a browser-based code editor, likely CodePen, displaying a demonstration of implicit grids. The page title is "Implicit grid" and the URL is <https://codepen.io/matuzo/pen/eyXdOa?editors=1100>. The editor interface includes tabs for HTML, CSS (SCSS), and JS, along with buttons for Save, Fork, Settings, Change View, and user profile.

The HTML pane shows the following code:

```
1 <div class="grid">
2   <div class="grid-item">1</div>
3   <div class="grid-item">2</div>
4   <div class="grid-item">3</div>
5   <div class="grid-item">4</div>
6   <div class="grid-item">5</div>
7 </div>
```

The CSS (SCSS) pane shows the following code:

```
.grid {
  display: grid;
  grid-auto-flow: column;
}
```

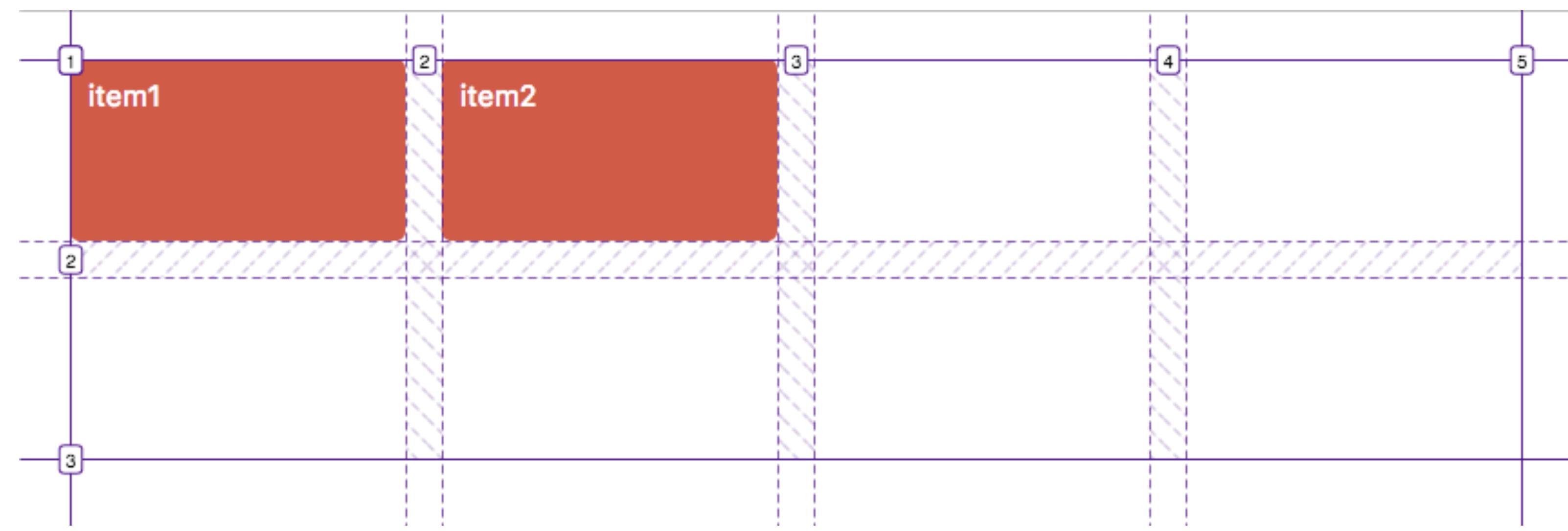
The preview area shows a horizontal row of five red rectangular boxes labeled 1 through 5 from left to right, demonstrating the implicit grid layout.

IMPLICIT GRIDS

bit.ly/grid_implicit

Explicit grids

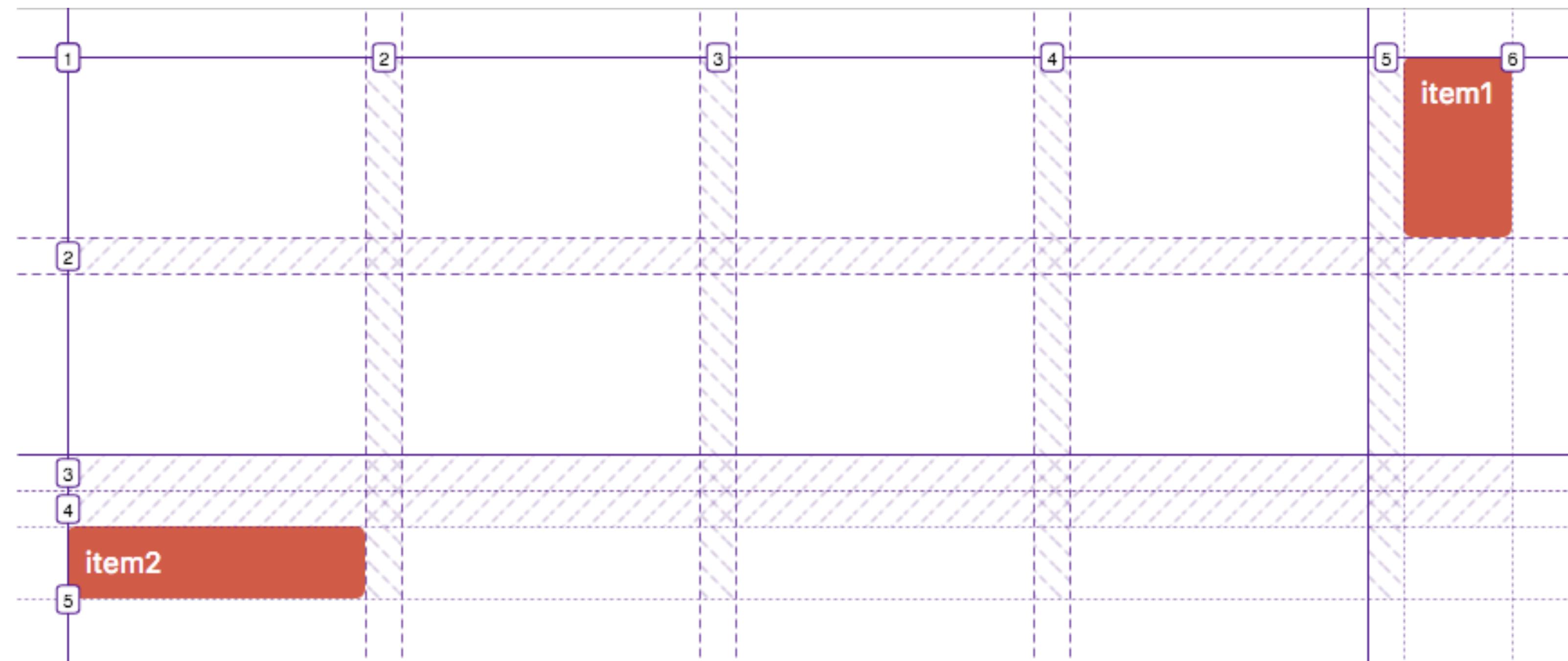
- We can define a fixed number of lines and tracks that form a grid by using the properties `grid-template-rows`, `grid-template-columns`, and `grid-template-areas`.
- This manually defined grid is called the explicit grid.



EXPLICIT GRID

Implicit grids

- The grid container automatically generates grid tracks by adding grid lines to the grid
 - ... if there are more grid items than cells in the grid
 - ...when a grid item is placed outside of the explicit grid.
- The widths and heights of the implicit tracks are set automatically. By default, they are only big enough to fit the placed grid items.



IMPLICIT GRIDS

The screenshot shows a CodePen editor window titled "Sizing implicit grids" by Manuel Matuzovic PRO. The page URL is <https://codepen.io/matuzo/pen/opVBBK?editors=1100>. The editor interface includes a toolbar with Save, Fork, Settings, and Change View buttons, and a sidebar with Console, Assets, Comments, and a file icon.

HTML

```
<div class="grid">
  <div class="grid-item">1</div>
  <div class="grid-item">2</div>
  <div class="grid-item">3</div>
  <div class="grid-item">4</div>
  <div class="grid-item">5</div>
</div>
```

CSS (SCSS)

```
.grid {
  display: grid;
  grid-template-columns: 200px 200px;
  grid-template-rows: 200px;
  grid-gap: 20px;
  grid-auto-rows: 80px;
}
```

JS

Last saved less than a minute ago [Edit](#) [Delete](#) [Share](#) [Export](#) [Embed](#) [Collections](#)

SIZING IMPLICIT GRIDS

bit.ly/grid_implicit2

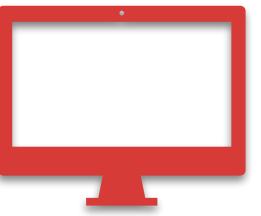
Grid Basics

Sizing implicit columns

```
.container {  
    grid-auto-columns: 80px;  
}
```

Sizing implicit rows

```
.container {  
    grid-auto-rows: 80px;  
}
```



EXERCISE 1

300px

Element 1

Element 3

200px

Element 2

Element 4

Gutter: 20px

Element 1

Element 3

100px high

Element 2

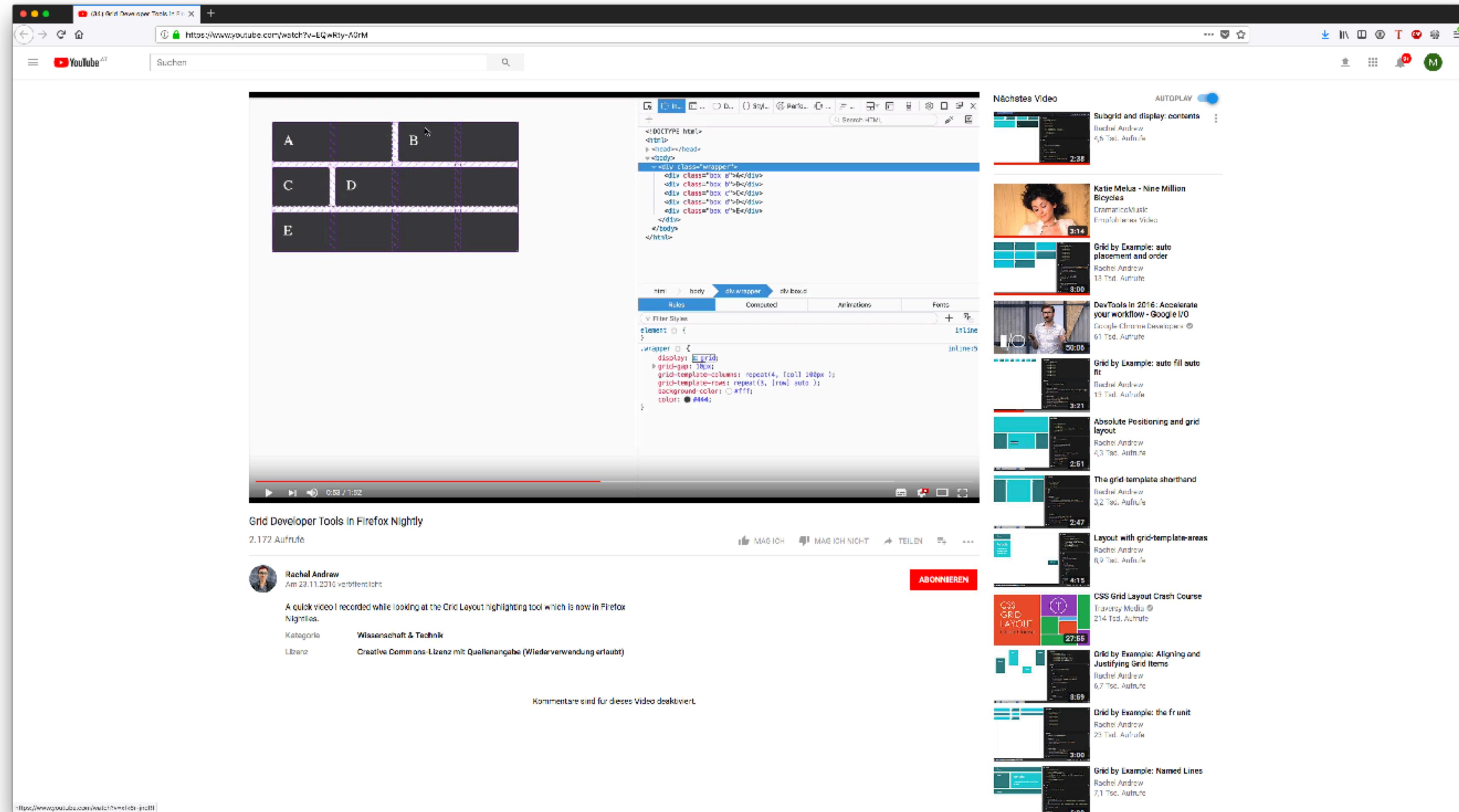
Element 4

DEBUGGING

Debugging

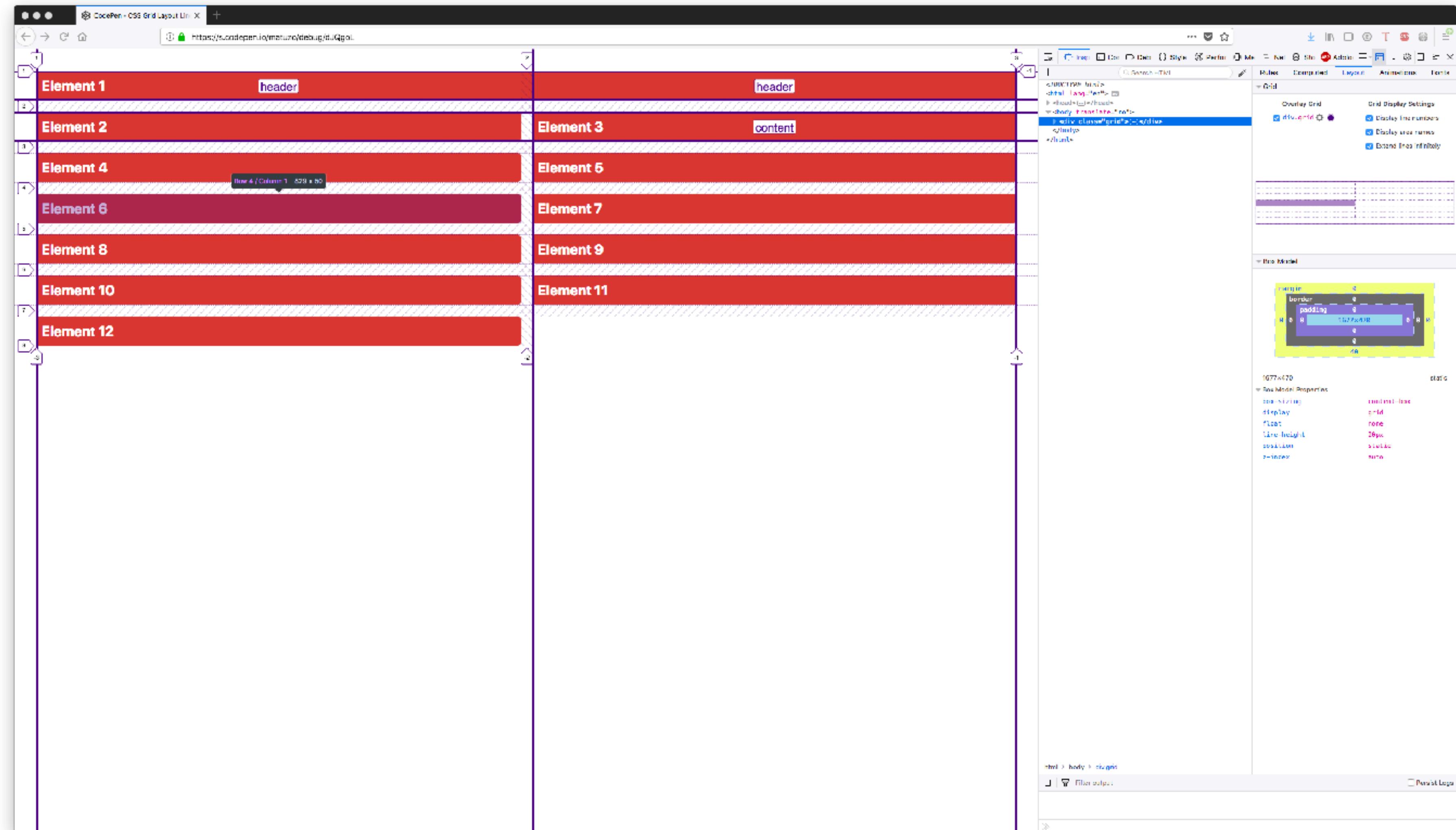
Firefox is currently the best browser for debugging.

[https://developer.mozilla.org/en-US/docs/Tools/Page_Inspector/
How_to/Examine_grid_layouts](https://developer.mozilla.org/en-US/docs/Tools/Page_Inspector/How_to/Examine_grid_layouts)



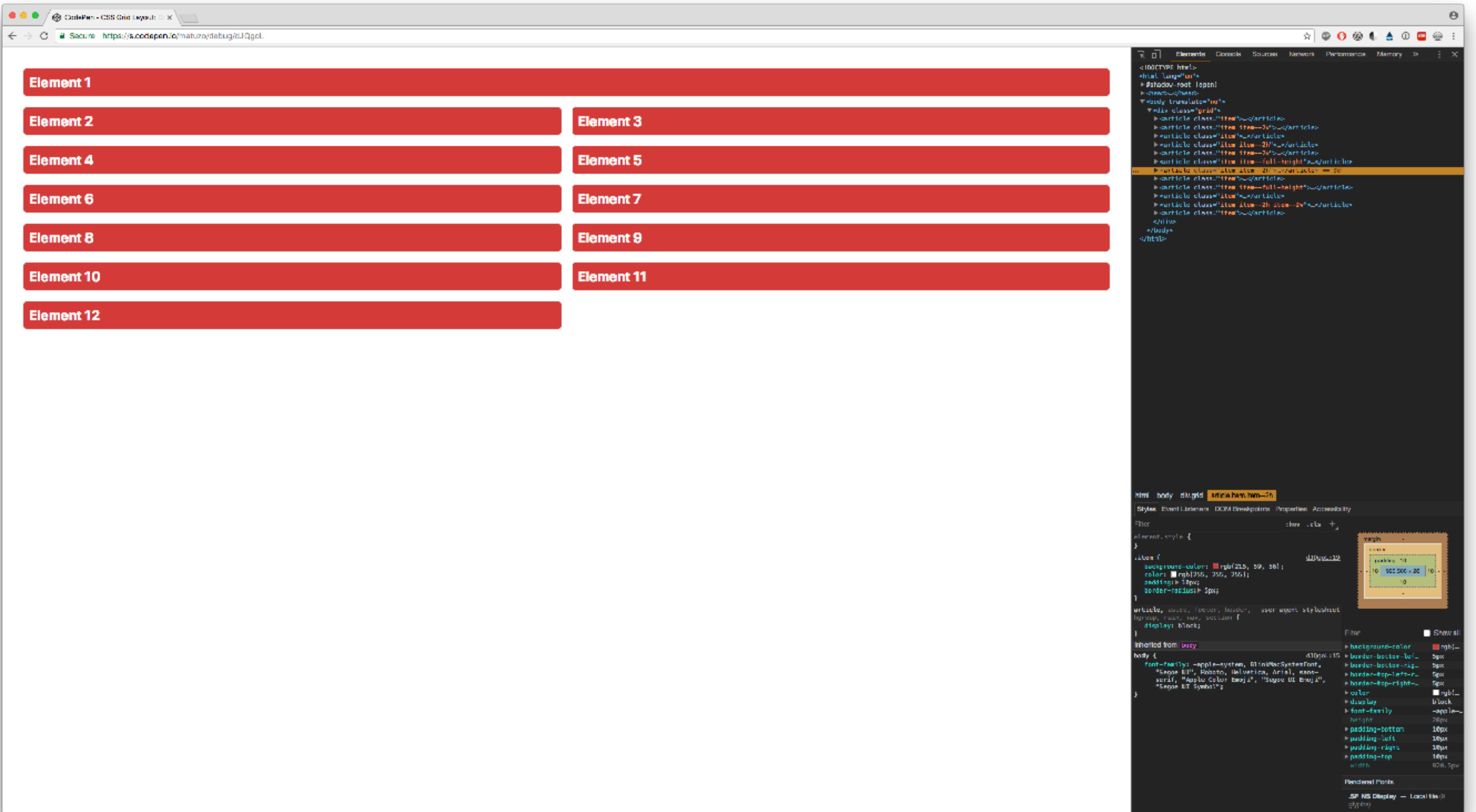
GRID DEBUGGING IN FIREFOX

www.youtube.com/watch?v=EQwRty-A0rM



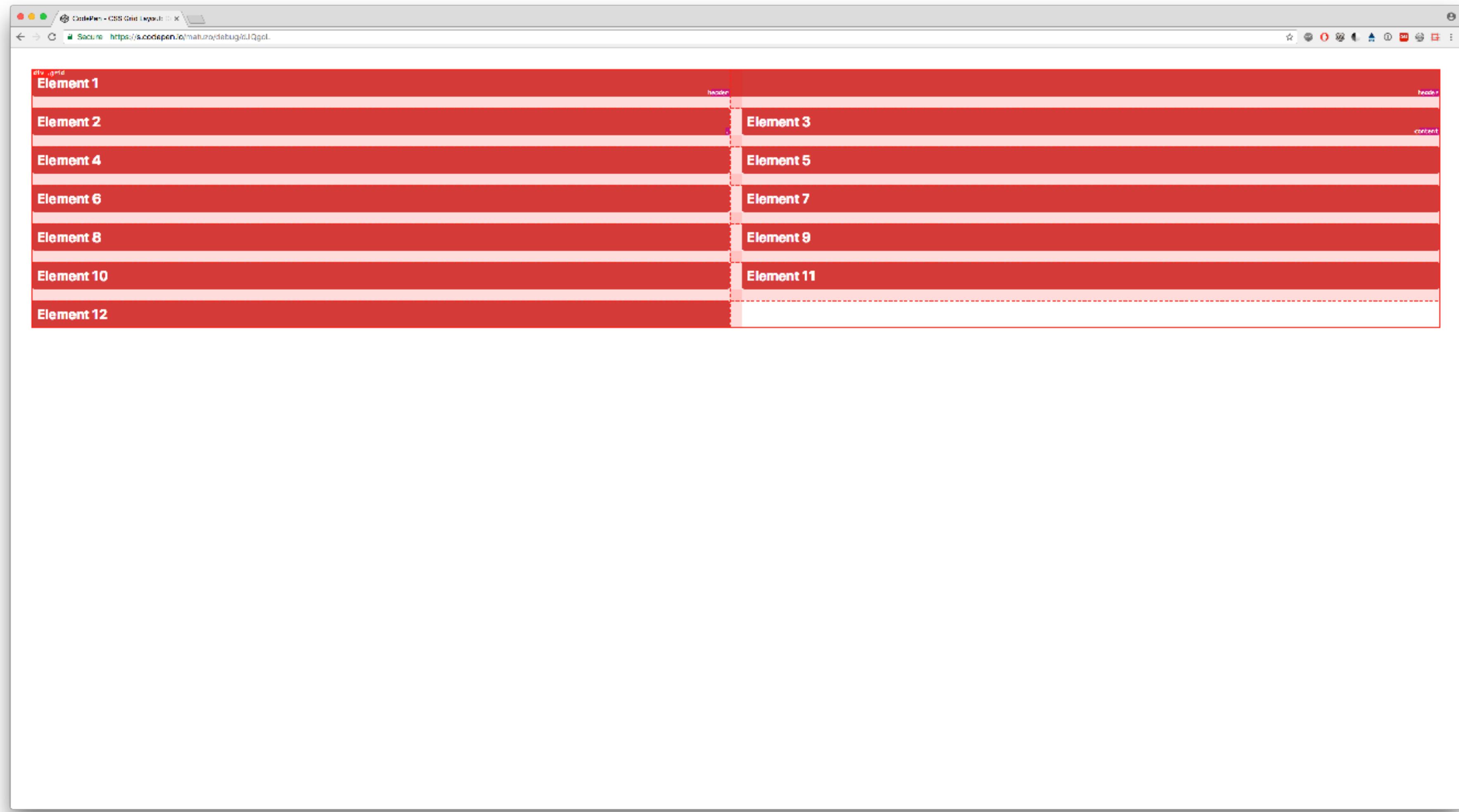
DEBUGGING IN FIREFOX

bit.ly/grid_debug



DEBUGGING IN CHROME

bit.ly/grid_debug



GRIDMAN - CSS GRID INSPECTOR. ULTRA FAST!

[Chrome](#)

GRID BASICS

fr unit, repeat, minmax

A screenshot of a CodePen interface titled "CSS Grid Layout Basics: fr unit". The page displays a 3x3 grid of red rectangular elements, each containing white text. The elements are labeled as follows:

- Row 1: Element 1, Element 2, Element 3
- Row 2: Element 4, Element 5, Element 6
- Row 3: Element 7, Element 8, Element 9

The grid is defined by the following CSS:

```
.grid {  
  display: grid;  
  grid-template-columns: 1fr 150px 2fr;  
  grid-template-rows: 1fr 50px 2fr;  
  grid-gap: 20px;  
}  
  
.grid > * {  
  height: 100%;  
}  
  
.grid > * {  
  border: 1px solid black;  
  padding: 10px;  
}
```

FR UNIT

bit.ly/grid_fr

Grid Basics

Flexible units (**fr**)

```
.container {  
    grid-template-columns: 1fr 150px 2fr;  
    grid-template-rows: 1fr 50px 2fr;  
}
```

The screenshot shows a web browser window with the title "CSS Grid Layout Basics: repeat". The page displays three examples of CSS Grid layout using the `repeat()` function.

HTML:

```
1 <div class="grid">
2   <article class="item">
3     <h2>Element 1</h2>
4   </article>
5   <article class="item">
6     <h2>Element 2</h2>
7   </article>
8   <article class="item">
9     <h2>Element 3</h2>
10  </article>
11 <article class="item">
12   <h2>Element 4</h2>
13 </article>
14 <article class="item">
15   <h2>Element 5</h2>
16 </article>
17 <article class="item">
18   <h2>Element 6</h2>
19 </article>
```

CSS:

```
1 .grid {
2   display: grid;
3   grid-template-columns: repeat(3, 200px);
4   grid-gap: 20px;
5   /* Repeating */
6 }
7
8 .grid2 {
9   /* Repeating a track listing */
10  grid-template-columns: repeat(2, 1fr 200px);
11 }
12
13 .grid3 {
14   /* Mixing units and lists */
15   grid-template-columns: 150px repeat(3, 1fr) 200px;
16 }
```

JS:

```
Last saved less than a minute ago
```

Toolbar:

- Save
- Fork
- Settings
- Change View

REPEATING TRACKS

bit.ly/grid_repeating

Grid Basics

Repeating tracks or track lists

```
.container {  
    grid-template-columns: repeat(3, 200px);  
    grid-template-columns: repeat(2, 1fr 200px);  
    grid-template-columns: 150px repeat(3, 1fr) 200px;  
}
```

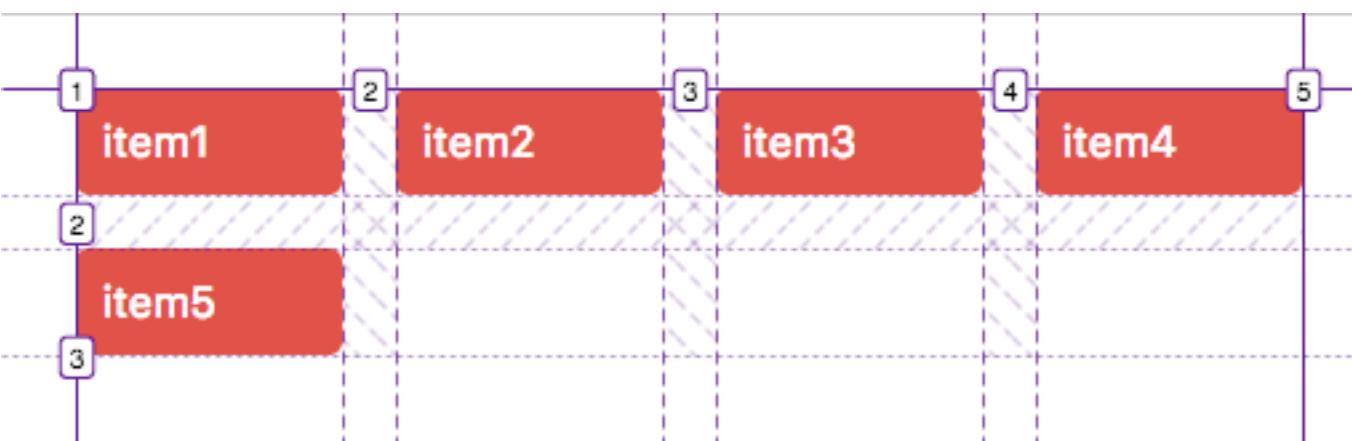
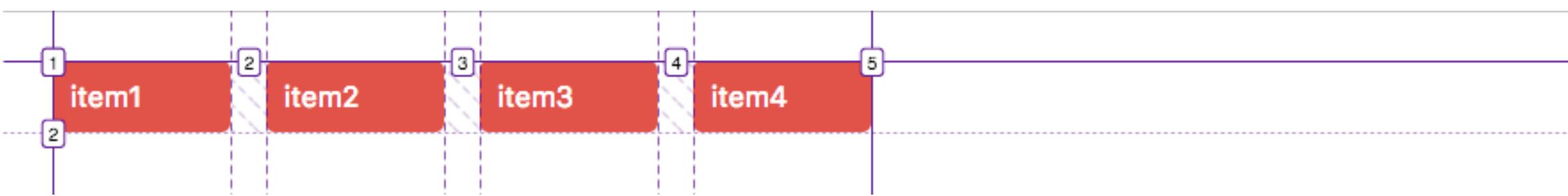
```
.grid {  
  grid-template-columns: repeat(4, 100px);  
}
```



```
.grid {  
  grid-template-columns: repeat(auto-fill, 100px);  
}
```



```
.grid {  
  grid-template-columns: repeat(auto-fit, 100px);  
}
```



Grid Basics

Auto-filling and auto-fitting tracks

```
.container {  
    grid-template-columns: repeat(auto-fill, 200px);  
    grid-template-columns: repeat(auto-fit, 100px);  
}
```

The screenshot shows a web-based code editor interface with a dark theme. On the left, there are two visual representations of a grid layout. The top representation is a 3x3 grid of red boxes labeled Element 1 through Element 9. The bottom representation is a 2x5 grid of red boxes labeled Element 1 through Element 8. The right side of the interface contains three panels: 'HTML', 'CSS', and 'JS'. The 'HTML' panel shows the DOM structure with articles and h2 elements. The 'CSS' panel contains the following CSS code:

```
grid {
  display: grid;
  grid-template-columns: repeat(3, 200px);
  grid-auto-rows: minmax(150px, auto);
  grid-gap: 20px;
}

.grid2 {
  grid-template-columns: 1fr minmax(400px, 1fr);
  grid-auto-rows: auto;
}
```

The 'JS' panel is currently empty.

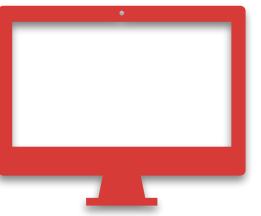
MINMAX

bit.ly/grid_minmax

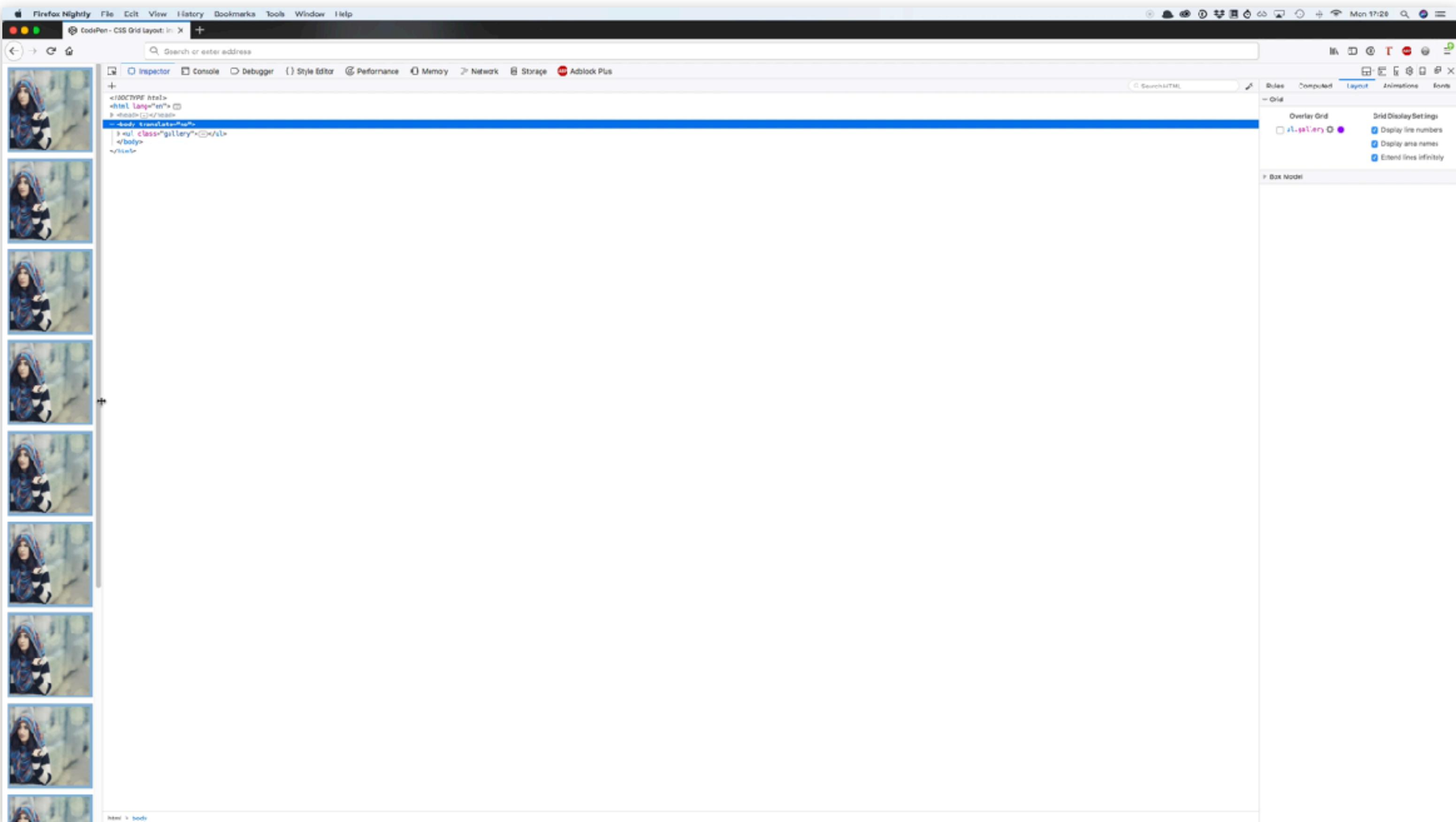
Grid Basics

Defining min and max values

```
.container {  
    grid-auto-rows: minmax(150px, 250px);  
    grid-template-columns: 1fr minmax(400px, 1fr);  
}
```



EXERCISE 2



[BIT.LY/GRID_EXERCISE2](https://bit.ly/grid_exercise2)

CSS BOX ALIGNMENT MODULE

Box Alignment Module

- Separate specification for alignment properties.
- justify-items, align-items, justify-self, align-self, justify-content, align-content, grid-gap
- place-items, place-self, gap
- Properties do not just apply to Grid and Flexbox (In the future also to block layout).

justify-content: space-evenly for flex-containers

`justify-content: space-evenly` works great with grid-containers, but support for flex-containers is currently (07/2017) very limited.

space-evenly
 Works in every major desktop browser except for Edge 16)

The alignment subjects are distributed so that the spacing between any two adjacent alignment subjects, before the first alignment subject, and after the last alignment subject is the same.



space-between

The first alignment subject is placed flush with the start edge of the alignment container, the last alignment subject is placed flush with the end edge of the alignment container, and the remaining alignment subjects are distributed so that the spacing between any two adjacent alignment subjects is the same.



space-around

The alignment subjects are evenly distributed in the alignment container, with a half-size space on either end. The alignment subjects are distributed so that the spacing between any two adjacent alignment subjects is the same, and the spacing before the first and after the last alignment subject is half the size of the other spacing.



HTML

```

1 <h1>justify-content: space-evenly for flex-containers</h1>
2
3 <p><code>justify-content: space-evenly</code> <a href="https://css-tricks.com/snippets/css/complete-guide-grid/#article-header-id-28">works great with grid-containers</a>, <del>but support for flex-containers is currently (time datetime="11-07-2017">07/2017</time>) very limited.</del></p>
4
5 <h2>space-evenly</h2>
6 (Works in every major desktop browser except for Edge 16)
7 <blockquote>The alignment subjects are distributed so that the spacing between any two adjacent alignment subjects, before the first alignment subject, and after the last alignment subject is the same.</blockquote>
8 <div class="parent evenly">
9 <div class="item">item 3</div>
10 <div class="item">item 3</div>
11 <div class="item">item 3</div>
12 </div>
13
14 <h2>space-between</h2>
15 <blockquote>
16 The first alignment subject is placed flush with the start edge of the alignment container, the last alignment subject is placed flush with the end edge of the alignment container, and the remaining alignment subjects are distributed so that the spacing between any two adjacent alignment subjects is the same.</blockquote>
17 <div class="parent between">
18 <div class="item">item 3</div>
19 <div class="item">item 3</div>
20 <div class="item">item 3</div>
21 </div>
22
23 <h2>space-around</h2>
24 <blockquote>The alignment subjects are evenly distributed in the alignment container, with a half-size space on either end. The alignment subjects are distributed so that the spacing between any two adjacent alignment subjects is the same, and the spacing before the first and after the last alignment subject is half the size of the other spacing.
  
```

CSS

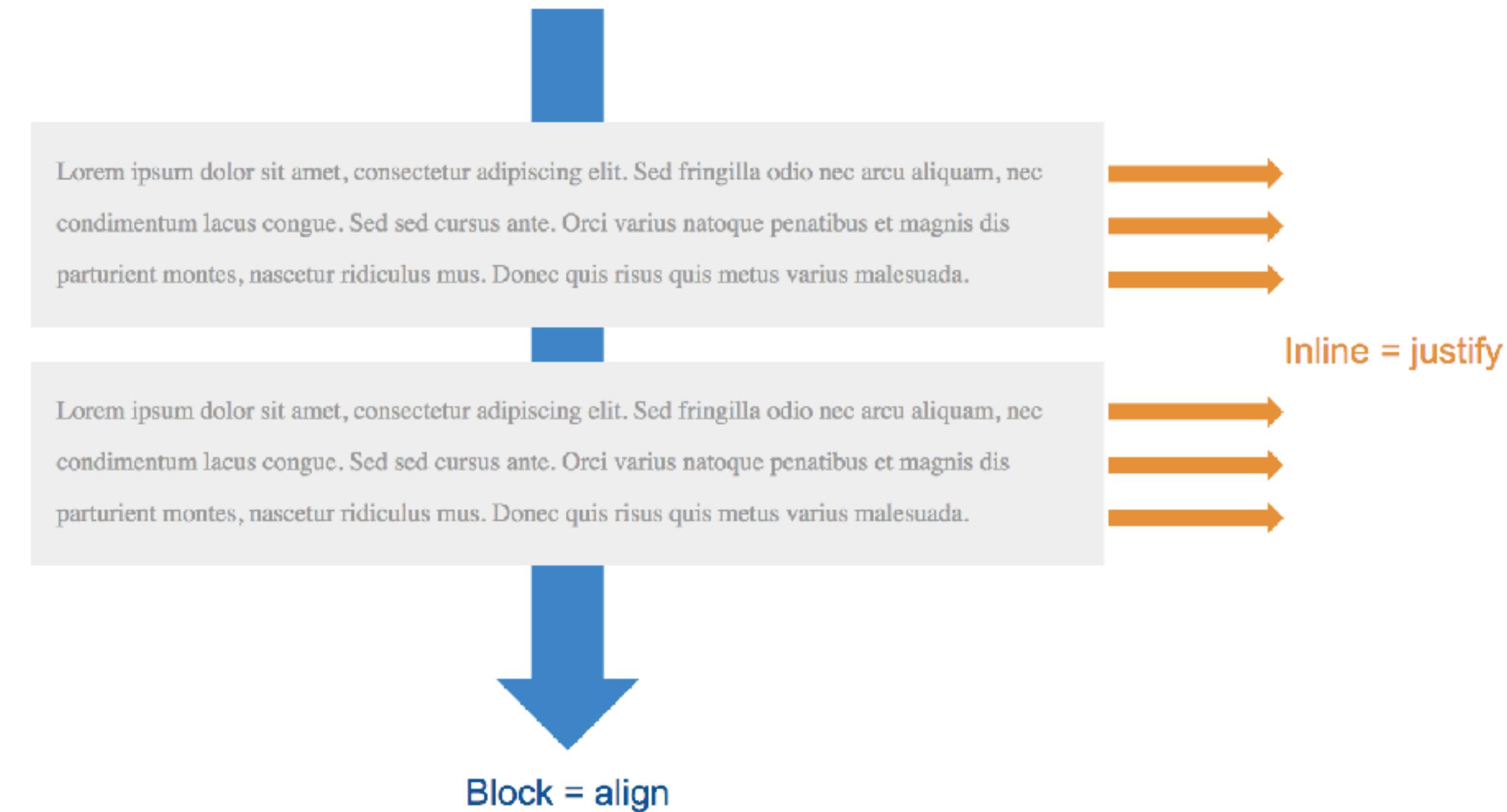
```

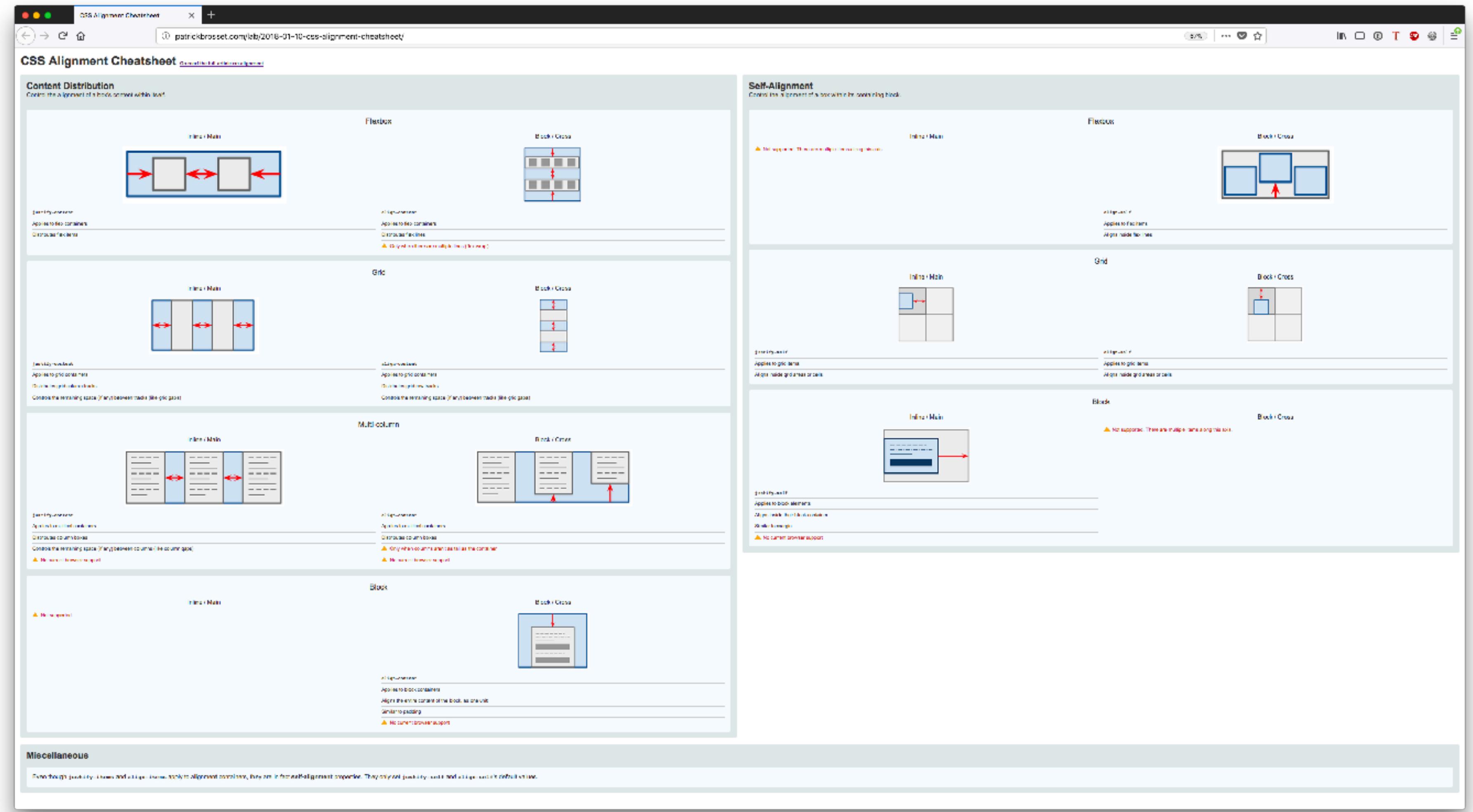
1 body {
2   max-width: 500px;
3   line-height: 1.4;
4 }
5
6 .parent {
7   display: flex;
8   border: 2px solid #000000;
9   margin-bottom: 30px;
10 padding: 20px 0;
11 }
12
13 .evenly {
14   justify-content: space-evenly;
15 }
16
17 .between {
18   justify-content: space-between;
19 }
20
  
```

JS

Box Alignment Module

- z-axis aside, we have two axes on which we can work on.
- The inline axis is the one each line of text runs on.
- The other one is the block axis and it's the one along which blocks are stacked.





ALIGNMENT CHEATSHEET

patrickbrosset.com/lab/2018-01-10-css-alignment-cheatsheet/

BOX ALIGNMENT

**justify-items, align-items, justify-self, align-self
justify-content, align-content**

Box Alignment

Inline tracks

```
.container {  
    justify-items: center;  
}
```

Block tracks

```
.container {  
    align-items: start;  
}
```

Values: **start**, **end**, **center**,
stretch (default)

The screenshot shows a CodePen interface with a dark theme. The title bar reads "CSS Grid Layout Box Alignment: justify-items, align-items, align-self, justify-self." Below the title, there are tabs for "HTML", "CSS", and "JS".

HTML:

```
<div>
  <article class="item">
    <h2>Element 4</h2>
  </article>
  <article class="item">
    <h2>Element 5</h2>
  </article>
</div>
```

CSS:

```
grid-auto-rows: 180px;
grid-gap: 20px;
}

.grid1 {
  justify-items: center;
  place-items: center;
}

.grid2 {
  align-items: end;
}

.grid3 .item:nth-child(1) {
  align-self: end;
}

.grid3 .item:nth-child(2) {
  align-self: center;
}

.grid3 .item:nth-child(3) {
  justify-self: center;
}

.grid3 .item:nth-child(4) {
  justify-self: start;
}

.grid3 .item:nth-child(5) {
  align-self: center;
  justify-self: center;
  place-self: center;
}
```

JS:

```
last saved 9 months ago
```

The preview area displays four different grid layouts using the CSS properties shown in the CSS panel.

- Grid 1:** A 2x4 grid where all items are centered both horizontally and vertically.
- Grid 2:** A 2x4 grid where all items are aligned to the bottom edge of their respective cells.
- Grid 3:** A 2x4 grid demonstrating item-specific alignment:
 - Items 1 and 2 are aligned to the bottom of their cells.
 - Item 3 is centered within its cell.
 - Item 4 is aligned to the top of its cell.
 - Item 5 is centered within its cell.
- Grid 4:** A 2x4 grid where all items are aligned to the top edge of their respective cells.

ALIGNING CELLS

bit.ly/grid_align

Box Alignment

Inline cell alignment

```
.grid-item {  
    justify-self: center;  
}
```

Block cell alignment

```
.grid-item {  
    align-self: start;  
}
```

Values: **start**, **end**, **center**,
stretch (default)

A screenshot of a CodePen interface displaying a CSS Grid layout example. The layout consists of 9 red rectangular boxes arranged in a 3x3 grid. The boxes are labeled Element 1 through Element 9. The grid is contained within a container with the class ".grid". The CSS code includes properties for the grid itself and for aligning the content within it.

```
HTML
<div class="grid">
  <article class="item">
    <h2>Element 1</h2>
  </article>
  <article class="item">
    <h2>Element 2</h2>
  </article>
  <article class="item">
    <h2>Element 3</h2>
  </article>
  <article class="item">
    <h2>Element 4</h2>
  </article>
  <article class="item">
    <h2>Element 5</h2>
  </article>
  <article class="item">
    <h2>Element 6</h2>
  </article>
  <article class="item">
    <h2>Element 7</h2>
  </article>
  <article class="item">
    <h2>Element 8</h2>
  </article>
  <article class="item">
    <h2>Element 9</h2>
  </article>
</div>
```

```
CSS
.grid {
  display: grid;
  grid-template-columns: repeat(3, 200px);
  grid-auto-rows: 100px;
  border: 2px solid #333;
  width: 800px;
  height: 600px;
}

.grid {
  justify-content: space-around;
  align-content: space-between;
}
```

JS

ALIGNING THE WHOLE GRID

bit.ly/grid_align2

Box Alignment

Inline grid positioning

```
.grid-item {  
    justify-content: center;  
}
```

Block positioning

```
.grid-item {  
    align-content: end;  
}
```

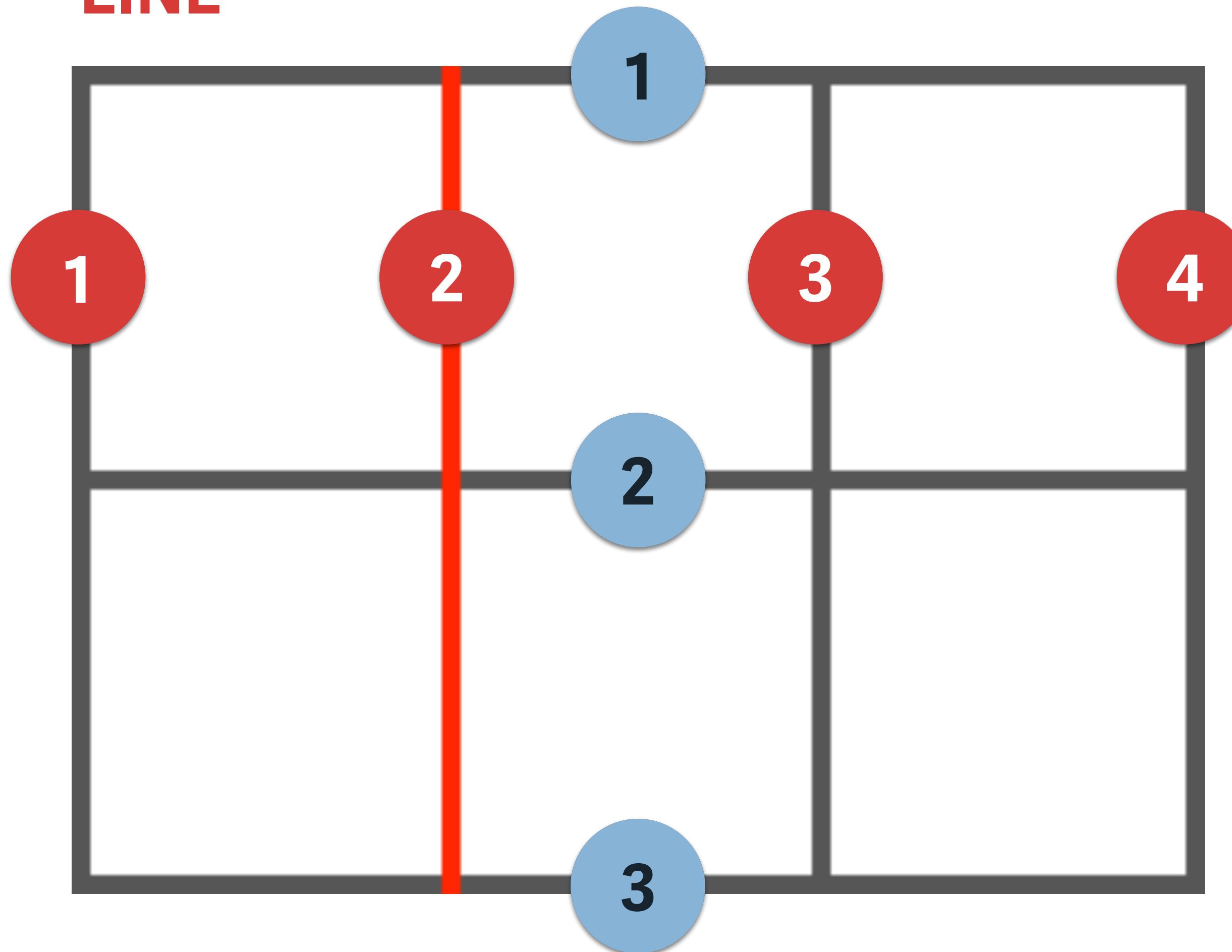
Values: **start** (default), **end**,
center, **space-between**,
space-around, **space-evenly**



**grid-column, grid-row, grid-column-start,
grid-column-end, grid-row-start, grid-row-end**

Terminology

LINE



Illustrations: <https://gridbyexample.com/what>

The screenshot shows a CSS Grid Layout editor interface with a grid of 12 red rectangular elements labeled Element 1 through Element 12. The grid is structured as follows:

- Row 1:** Element 1 (spanning 4 columns), Element 3 (1 column), Element 2 (1 column).
- Row 2:** Element 4 (1 column), Element 5 (1 column), Element 6 (1 column).
- Row 3:** Element 7 (spanning 4 columns), Element 8 (1 column), Element 9 (1 column), Element 10 (spanning 2 columns), Element 12 (1 column).

The CSS code in the editor is:

```
<div class="grid">
  <article class="item">
    <h2>Element 1</h2>
  </article>
  <article class="item">
    <h2>Element 2</h2>
  </article>
  <article class="item">
    <h2>Element 3</h2>
  </article>
  <article class="item">
    <h2>Element 4</h2>
  </article>
  <article class="item">
    <h2>Element 5</h2>
  </article>
  <article class="item">
    <h2>Element 6</h2>
  </article>
  <article class="item">
    <h2>Element 7</h2>
  </article>
  <article class="item">
    <h2>Element 8</h2>
  </article>
  <article class="item">
    <h2>Element 9</h2>
  </article>
  <article class="item">
    <h2>Element 10</h2>
  </article>
  <article class="item">
    <h2>Element 11</h2>
  </article>
  <article class="item">
    <h2>Element 12</h2>
  </article>
</div>
```

At the bottom of the editor, there are tabs for "Console", "Assets", "Commands", and "JS".

PLACING CELLS ON LINES

bit.ly/grid_lines

Lines

Cell start and end point.
(vertical track)

Value: Line number

Shorthand:

```
.item {  
    grid-column-start: 2;  
    grid-column-end: 4;  
}
```

```
.item {  
    grid-column: 2 / 4;  
}
```

Lines

Cell start and end point.
(horizontal track)

Value: Line number

Shorthand:

```
.item {  
    grid-row-start: 1;  
    grid-row-end: 3;  
}
```

```
.item {  
    grid-row: 1 / 3;  
}
```

A screenshot of a CodePen editor window titled "Lower grid-column-end value than grid-column-start". The window shows a preview of a grid layout with two items, "Item1" and "Item2", and the underlying HTML and CSS code.

Grid

```
<h2>Grid</h2>
<section class="grid">
  <div class="item">Item1</div>
  <div class="item">Item2</div>
</section>
```

HTML

```
1 <h2>Grid</h2>
2 <section class="grid">
3   <div class="item">Item1</div>
4   <div class="item">Item2</div>
5 </section>
```

CSS (SCSS)

```
1 .grid {
2   display: grid;
3   grid-template-columns: 1fr 1fr 1fr 1fr;
4   grid-template-rows: 100px 100px;
5   grid-gap: 20px;
6 }
7
8
9 .item:first-child {
10   grid-column-end: 2;
11   grid-column-start: 4;
12
13
14 // Some as:
15 // grid-column-start: 2;
16 // grid-column-end: 4;
17 }
```

JS

```
Last saved less than a minute ago
```

LOWER END VALUE THAN START VALUE

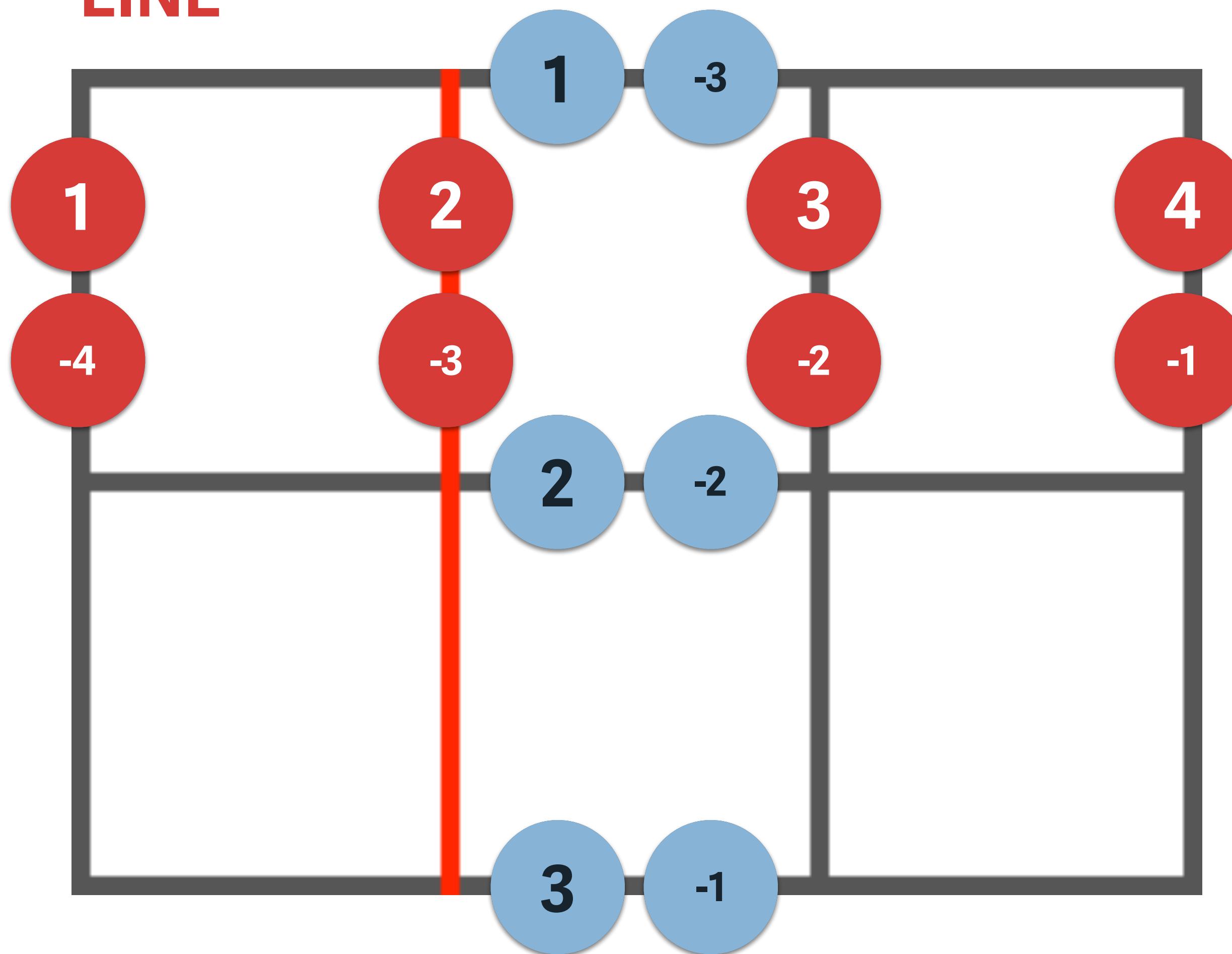
bit.ly/grid_lines7

Negative lines values

- Each line is represented by a positive and a negative number.
- For example, grid-column: 2 / -1; spans from the second line to the last line.
- It's possible to use negative values in both grid-row/column-start and grid-row/column-end.

Terminology

LINE



A screenshot of a CodePen editor window displaying a grid layout example. The page title is "Grid item from first to second to last". The editor interface includes tabs for "HTML", "CSS", and "JS".

HTML

```
<div class="grid">
  <article class="item">
    <h2>Element 1</h2>
  </article>
  <article class="item">
    <h2>Element 2</h2>
  </article>
  <article class="item">
    <h2>Element 3</h2>
  </article>
</div>
```

CSS

```
.grid {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-gap: 20px;
}

.item:first-child {
  grid-column: 1 / -1;
}

.item:nth-child(2) {
  grid-column: 1 / -2;
}

.item:nth-child(3) {
  grid-column: 1 / -3;
}
```

The preview area shows three red rectangular elements labeled "Element 1", "Element 2", and "Element 3" arranged horizontally. Element 1 spans all three columns, Element 2 spans two columns, and Element 3 spans one column.

NEGATIVE LINE VALUES

bit.ly/grid_lines5

A screenshot of a web-based code editor (CodePen) displaying a grid layout example. The interface includes tabs for HTML, CSS, and JS, along with a preview area and various editing tools.

HTML

```
<div class="grid">
  <article class="item">
    <h2>Element 1</h2>
  </article>
  <article class="item">
    <h2>Element 2</h2>
  </article>
  <article class="item">
    <h2>Element 3</h2>
  </article>
</div>
```

CSS (less)

```
.grid {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-template-rows: repeat(3, 100px);
  grid-gap: 20px;
}

.item:first-child {
  grid-column-start: -3;
  grid-row: -2;
}

.item:nth-child(2) {
  grid-column: -2;
  grid-row: -3;
}

.item:nth-child(3) {
  grid-column: -4;
  grid-row: -4;
}
```

The preview area shows three red rectangular elements labeled "Element 1", "Element 2", and "Element 3". Element 1 is at the top-left, Element 2 is at the middle-right, and Element 3 is at the bottom-left. They are arranged in a 3x3 grid with gaps between them.

NEGATIVE LINE VALUES

bit.ly/grid_lines6

span Keyword

- A grid item spans a single cell by default.
- That can be changed by applying the `span` keyword.
- For example, `grid-column-start: 1` and `grid-column-end: span 2` will make the item span two cells, from the first to the third line.
- The `span` keyword may also be used on start properties.
- For example, with `grid-column-end: -1` and `grid-column-start: span 2` the item will be placed at the last line and span 2 cells, from the last to third to last line.

The screenshot shows a CodePen editor window titled "CSS Grid Layout: span keyword". The preview area displays a grid layout with four items: "Item1" (top-left), "Item2" (bottom-left), "Item2" (bottom-right), and "Item2" (bottom-right). The CSS panel contains the following code:

```
grid {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr 1fr;
  grid-template-rows: 100px 100px;
  grid-auto-columns: 40px;
  grid-gap: 20px;
}

.item:nth-child(1) {
  grid-column: 1 / span 3;
}

.item:nth-child(2) {
  grid-column-start: span 2;
  grid-column-end: 3;
}

.item:nth-child(3) {
  grid-column-end: -1;
  grid-column-start: span 2;
}
```

THE SPAN KEYWORD

bit.ly/grid_lines8

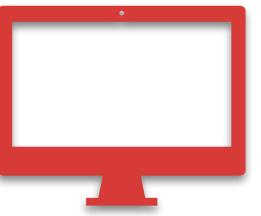
Lines

Span keyword

```
.item {  
    grid-row: 3 / span 2;  
    grid-column: span 2;  
}
```

Spanning from line **n** to the last line.

```
.item {  
    grid-column: 2 / -1;  
}
```



EXERCISE 3

Element 2

Element 1

Element 3

Element 5

Element 7

Element 9

Element 6

Element 8

Element 4

A screenshot of a web-based CSS Grid layout editor. The interface includes a header with a back button, forward button, refresh button, and a URL bar showing <https://codepen.io/matzko/ben/Omymo4/editors-1100>. Below the header is a title bar with a logo, the text "CSS Grid Layout Lines: grid-auto-flow: dense", and a "Fork" button. The main area contains a grid of 12 red rectangular elements labeled Element 1 through Element 12. The grid has 4 columns and 3 rows. Elements 1-4 are in the first row, 5-8 in the second, and 9-12 in the third. Elements 1, 2, 3, 5, 6, 7, 8, 9, and 11 have a height of 100px, while 4, 10, and 12 have a height of 200px. The right side of the editor shows three panels: "HTML" containing the DOM structure, "CSS" containing the CSS styles, and "JS" which is empty. The CSS panel shows the following code:

```
grid-template-columns: repeat(4, 1fr);
grid-auto-rows: 100px;
grid-gap: 20px;
```

```
.grid {
  display: grid;
  grid-template-columns: repeat(4, 1fr);
  grid-auto-rows: 100px;
  grid-gap: 20px;
}

.item--2v {
  grid-row-end: span 2;
}

.item--2h {
  grid-column-end: span 2;
}

.item--full-height {
  grid-row-end: span 4;
}
```

DENSE PACKING MODE

bit.ly/grid_dense

The screenshot shows a web browser window with the URL <https://codepen.io/matuzo/pen/QvmpqJ?editors=1100>. The page title is "CSS Grid Layout Lines: named lines". The layout consists of a header, three articles, two ads, and a sidebar with a "Related Posts" section and a "More Posts" section. The sidebar also contains a "partners" section with a single item. The code editor on the right shows the corresponding HTML, CSS, and JS code.

HTML

```
37 <div class="item post">
38   <h2>More Posts</h2>
39   <ul>
40     <li>Element 1</li>
41     <li>Element 2</li>
42     <li>Element 3</li>
43     <li>Element 4</li>
44     <li>Element 5</li>
45     <li>Element 6</li>
46   </ul>
47 </div>
48 </aside>
49 <aside class="partners">
50   <div class="item partner">
51     <h2>Ad 1</h2>
52   </div>
53 </aside>
```

CSS

```
1 .grid {
2   display: grid;
3   grid-template-columns: 1fr 200px 200px;
4   grid-gap: 20px;
5   grid-auto-flow: dense;
6   align-items: start;
7 }
8
9 .header {
10   background-color: #17242D;
11   grid-column-start: 1;
12   grid-column-end: -1;
13 }
14
15 .article {
16   grid-column-start: 1;
17 }
18
19 .posts {
20   grid-column-start: 3;
21   grid-row-end: span 3;
22 }
23
24 .partners {
25   grid-column-start: 2;
26 }
27
28 .post {
29   background-color: #87B3D6;
30   margin-bottom: 20px;
31 }
32
33 .partner {
34   background-color: #7CC9BB;
35   margin-bottom: 20px;
36 }
```

JS

```
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
```

NAMING LINES

bit.ly/grid_lines2

Lines

Naming lines

```
.container {  
    grid-template-columns:  
        [line1] 20px [line2] 20px [line3] 20px [line4];  
}  
  
.grid-item {  
    grid-column: line2 / line4;  
}
```

The screenshot shows a web-based CSS Grid editor interface. On the left, there's a preview area displaying a 3x4 grid of 12 items, each labeled with its element number. Item 2 is a dark gray rectangle, while the other 11 items are solid red rectangles. To the right of the preview are three code editors: HTML, CSS, and JS. The HTML editor contains a structure of articles and sections. The CSS editor contains a grid template definition and a rule for every second item. The JS editor is currently empty.

```
HTML
1 <div class="grid">
2   <article class="item">
3     <h2>Element 1</h2>
4   </article>
5   <article class="item item--2v">
6     <h2>Element 2</h2>
7   </article>
8   <article class="item">
9     <h2>Element 3</h2>
10  </article>
11  <article class="item item--2h">
12    <h2>Element 4</h2>
13  </article>
14  <article class="item item--2v">
15    <h2>Element 5</h2>
16  </article>
17  <article class="item item--full-height">
18    <h2>Element 6</h2>
19 </article>
20 <article class="item">
21   <h2>Element 7</h2>
22 </article>
23 <article class="item item--2h">
24   <h2>Element 8</h2>
25 </article>
26 <article class="item item--2v">
27   <h2>Element 9</h2>
28 </article>
29 <article class="item item--2h">
30   <h2>Element 10</h2>
31 </article>
32 <article class="item item--2v">
33   <h2>Element 11</h2>
34 </article>
35 <article class="item item--full-height">
36   <h2>Element 12</h2>
37 </article>
38 </div>
```

```
CSS (css)
1 .grid {
2   display: grid;
3   grid-template-columns: repeat(4, 1fr);
4   grid-template-rows: [start] repeat(3, 200px [row]) [end];
5   grid-gap: 20px;
6 }
7
8 .item:nth-child(2) {
9   background-color: #17242D;
10  grid-row-start: row 2;
11 }
```

```
JS (js)
```

REPEATED NAMED LINES

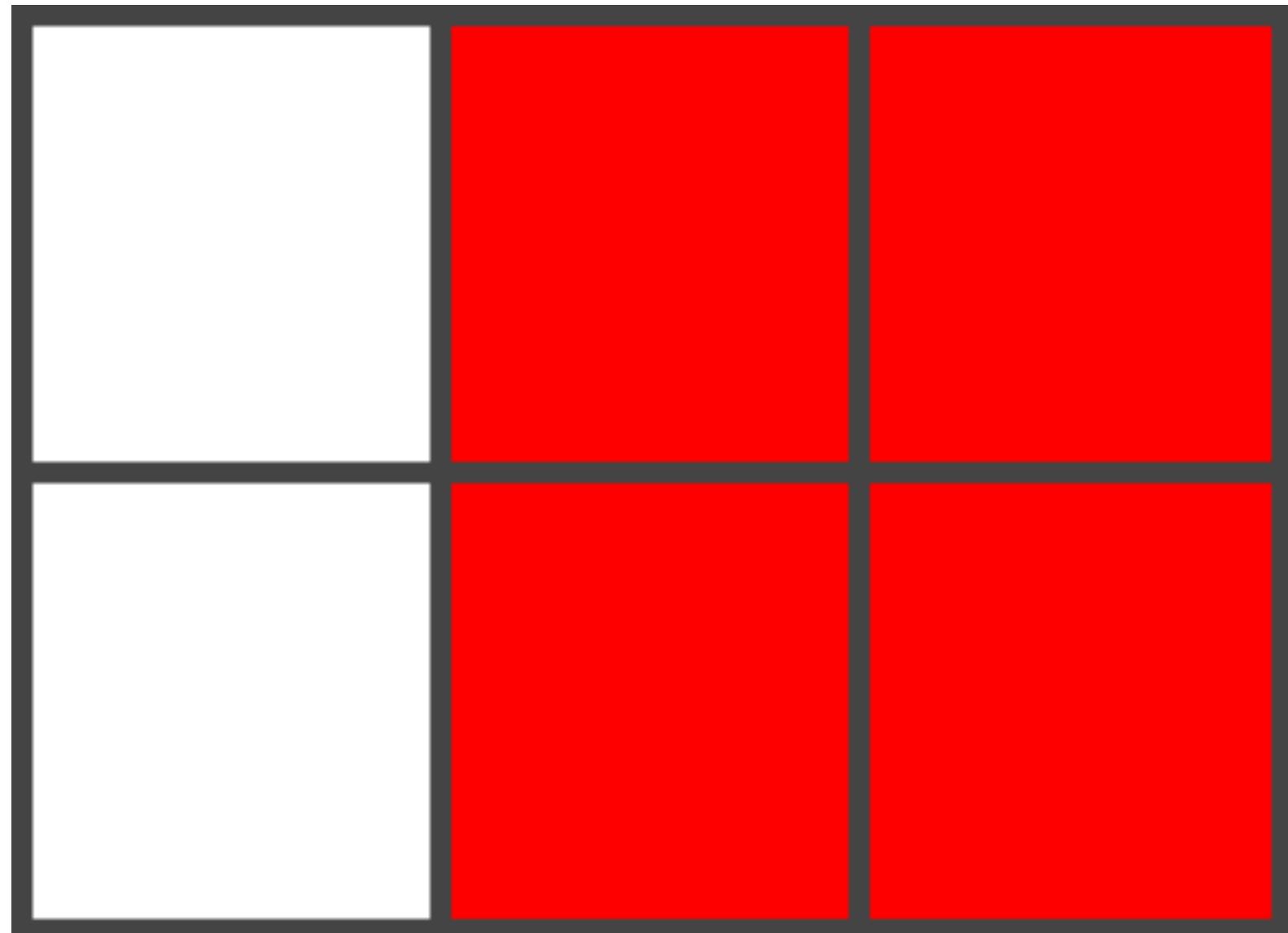
bit.ly/grid_lines3

GRID AREAS

`grid-area`

Terminology

AREA



One or more cells

Illustrations: <https://gridbyexample.com/what>

Areas

Defining areas

```
section {  
    grid-area: main;  
}
```

```
aside {  
    grid-area: sidebar;  
}
```

Areas

Defining layouts with areas

```
.grid {  
  grid-template-areas: "main sidebar";  
  grid-template-columns: 1fr 2fr;  
}
```

```
section {  
  grid-area: main;  
}
```

```
aside {  
  grid-area: sidebar;  
}
```

Areas

Defining layouts with areas

```
.grid {  
  grid-template-areas: "content sidebar"  
                      "content sidebar"  
                      "content sidebar";  
}
```

Areas

Defining layouts with areas

```
.grid {  
  grid-template-areas: "header header"  
                      "content sidebar"  
                      "footer footer";  
}
```

Areas

Skipping areas

```
.grid {  
  grid-template-areas: "header header"  
                      "content sidebar"  
                      ". footer";  
}
```

The screenshot shows a web browser window with a dark-themed CSS Grid Layout editor. The main content area displays a website layout with three red boxes labeled "Article 1", "Article 2", and "Article 3". To the right of these articles are two teal boxes labeled "Ad 1" and "Ad 2". Below the articles is a blue box labeled "Related Posts" containing a list of six items. Further down is another blue box labeled "More Posts" with a similar list. The right side of the editor features a vertical sidebar with tabs for "HTML", "CSS (scss)", and "JS". The "HTML" tab shows the structural code with grid areas like "articles", "posts", and "partners". The "CSS (scss)" tab contains the corresponding SCSS styles for these areas, including background colors and margin-bottom properties. The "JS" tab is currently empty.

```
grid-area: articles;
grid-area: posts;
grid-area: partners;
```

```
.article {
  margin-bottom: 20px;
}

.posts {
  margin-bottom: 20px;
}

.partners {
  margin-bottom: 20px;
}

.post {
  background-color: #8783D6;
  margin-bottom: 20px;
}

.partner {
  background-color: #70C9BB;
  margin-bottom: 20px;
}
```

GRID-AREA LAYOUTS

bit.ly/grid_area2

Areas

“Magic lines”.

By defining areas we get automatically named lines for free.

[AREA NAME] + **-start** or **-end** suffix.

```
.grid {  
  grid-template-areas: "header header"  
                      "content sidebar"  
                      "footer footer";  
}  
  
.grid-item {  
  grid-row-start: content-end;  
  grid-column-start: sidebar-start;  
}
```

The screenshot shows a web browser window with a dark-themed CSS Grid Layout editor. The layout consists of several sections:

- Header:** "My Website" title bar.
- Content Area:** Contains three red boxes labeled "Article 1", "Article 2", and "Article 3". Each article box contains placeholder text: "Lorem ipsum dolor sit amet, consectetur adipisicing elit. Quam modi nihil, alias dolores autem officia debitis unde nemo vel temporibus, tempora consequatur, culpa illum ad repudiandae preferendis? Porro, voluptatibus, cupiditate!"
- Right Sidebar:** Contains two teal boxes labeled "Ad 1" and "Ad 2".
- Related Posts:** A blue box containing a list of six items: "Element 1", "Element 2", "Element 3", "Element 4", "Element 5", and "Element 6".
- More Posts:** A blue box containing a list of six items: "Element 1", "Element 2", "Element 3", "Element 4", "Element 5", and "Element 6".

The right side of the interface features a code editor with three tabs: "HTML", "CSS (scss)", and "JS". The "HTML" tab shows the structure of the website, while the "CSS (scss)" tab displays the SCSS code for styling:

```
grid-area: footer;
background-color: #17242D;
grid-area: articles;
grid-area: posts;
grid-area: partners;
```

The "JS" tab is currently empty.

GRID-AREA MAGIC LINES

bit.ly/grid_area3

Areas

“Magic areas”.

By defining named lines we get automatically areas for free.

http://bit.ly/grid_area

```
.grid {  
  display: grid;  
  grid-template-columns: 2fr [aside-start] 1fr [aside-end];  
  grid-gap: 16px;  
}  
  
.grid-item:last-child {  
  background: blue;  
  grid-area: aside;  
}
```

A screenshot of a CSS Grid Layout editor interface, likely from CodePen, demonstrating the use of the `grid-area` property with magic areas.

The layout consists of a 2x2 grid of boxes:

- Top-left: A large blue box.
- Top-right: Two stacked red boxes.
- Bottom-left: A large red box.
- Bottom-right: A single red box.

The CSS code defines the grid structure and styles:

```
grid-template-columns: [content-start] 2fr [content-end aside-start] 1fr [aside-end];
grid-template-rows: [content-start] 100px [aside-start] 200px [content-end] 200px [aside-end];
```

```
.grid {
  display: grid;
  grid-template-columns: [content-start] 2fr [content-end aside-start] 1fr [aside-end];
  grid-template-rows: [content-start] 100px [aside-start] 200px [content-end] 200px [aside-end];
}

.grid-item:last-child {
  background: #87306;
  grid-area: content;
}
```

GRID-AREA MAGIC AREAS

bit.ly/grid_area

Areas

Using **grid-area** as a shorthand.

grid-row-start
grid-column-start
grid-row-end
grid-column-end

```
.grid-item {  
    grid-area: 4 / 2 / 5 / 3;  
}  
  
.grid-item {  
    grid-area: 2 / 2 / span 2 / span 2;  
}
```

The screenshot shows a web-based CSS editor interface for creating grid layouts. On the left, a preview area displays a 3x3 grid of colored boxes labeled Element 1 through Element 9. The first two columns are red, and the third column is green. Elements 1, 4, and 5 are in the top row; 2, 6, and 7 are in the middle row; and 3, 8, and 9 are in the bottom row. To the right of the preview are three code editors: HTML, CSS, and JS. The HTML editor shows the structure of the grid using the <grid> element and <article> elements. The CSS editor contains the following code:

```
.grid {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-auto-rows: 100px;
  grid-gap: 20px;
}

.item:first-child {
  grid-area: 4 / 2 / 5 / 3;
  // row-start / column-start / row-end / column-end
}

.item:nth-child(2) {
  background-color: #7CC0BB;
  grid-area: 2 / 2 / span 2 / span 2;
}
```

GRID-AREA SHORTHAND

bit.ly/grid_area1

SHORTHANDS

grid, grid-template

Shorthands

grid-template shorthand

row track-listing /
column track-listing

```
.grid {  
    grid-template-columns: 1fr 600px 1fr;  
    grid-template-rows: 200px auto 80px;  
}  
  
.grid {  
    grid-template: 200px auto 80px / 1fr 600px 1fr;  
}
```

Shorthands

grid-template shorthand
with areas

Areas + row tracks /
column track-listing

```
.grid {  
  grid-template-columns: 1fr 600px 1fr;  
  grid-template-rows: 200px auto 80px;  
  grid-template-areas: "header header header"  
                      ". content sidebar"  
                      "footer footer footer";  
}  
  
.grid {  
  grid-template: "header header header" 200px  
                ". content sidebar"  
                "footer footer footer" 80px /  
                1fr 600px 1fr;  
}
```

The screenshot shows a web-based code editor for CSS Grid layout. The interface includes a header with tabs for 'HTML', 'CSS (SCSS)', and 'JS', and a toolbar with buttons for 'Save', 'Fork', 'Settings', and 'Change View'. The main area displays a 3x3 grid of red boxes labeled 'Element 1' through 'Element 9'. A single green box labeled 'Element 2' spans two columns. The 'HTML' tab shows the following code:

```
<div class="grid">
  <article class="item">
    <h2>Element 1</h2>
  </article>
  <article class="item">
    <h2>Element 2</h2>
  </article>
  <article class="item">
    <h2>Element 3</h2>
  </article>
  <article class="item">
    <h2>Element 4</h2>
  </article>
  <article class="item">
    <h2>Element 5</h2>
  </article>
  <article class="item">
    <h2>Element 6</h2>
  </article>
  <article class="item">
    <h2>Element 7</h2>
  </article>
  <article class="item">
    <h2>Element 8</h2>
  </article>
  <article class="item">
    <h2>Element 9</h2>
  </article>
</div>
```

The 'CSS (SCSS)' tab shows the following SCSS code:

```
.grid {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-auto-rows: 100px;
  grid-gap: 20px;
}

.item:first-child {
  grid-area: 4 / 2 / 5 / 3;
  // row-start / column-start / row-end / column-end
}

.item:nth-child(2) {
  background-color: #7CC0BB;
  grid-area: 2 / 2 / span 2 / span 2;
}
```

GRID-TEMPLATE SHORTHAND

bit.ly/grid_short4

Shorthands

grid shorthand

grid-template-areas
grid-template-rows /
grid-template-columns,
grid-auto-flow,
grid-auto-rows /
grid-auto-columns

```
.grid {  
    grid-template-rows: 200px auto;  
    grid-template-columns: 1fr auto 1fr;  
}
```

```
.grid {  
    grid: 200px auto / 1fr auto 1fr;  
}
```

Shorthands

grid shorthand

grid-template-areas
grid-template-rows /
grid-template-columns,
grid-auto-flow,
grid-auto-rows /
grid-auto-columns

```
.grid {  
    display: grid;  
    grid-template-rows: 200px 100px;  
    grid-auto-flow: column;  
    grid-auto-columns: 150px;  
}  
  
.grid {  
    grid: 200px 100px / auto-flow 150px;  
}
```

Shorthands

grid shorthand

grid-template-areas
grid-template-rows /
grid-template-columns,
grid-auto-flow,
grid-auto-rows /
grid-auto-columns

```
.grid {  
    display: grid;  
    grid-template-columns: 1fr 2fr;  
    grid-auto-flow: row;  
    grid-auto-rows: 200px;  
}  
  
.grid {  
    grid: auto-flow 200px / 1fr 2fr;  
}
```

A screenshot of a CodePen editor window titled "grid shorthand". The editor interface includes a header with tabs for "HTML", "CSS", and "JS", and a footer with buttons for "Save", "Fork", "Settings", and "Change View". The preview area shows a 2x3 grid of five items. Item 1 is in the top-left, item 2 is in the top-right, item 3 is in the bottom-left, item 4 is in the bottom-right, and item 5 is a red box at the bottom-left corner.

HTML

```
<div class="grid">
  <div class="grid_item">1</div>
  <div class="grid_item">2</div>
  <div class="grid_item">3</div>
  <div class="grid_item">4</div>
  <div class="grid_item">5</div>
</div>
```

CSS

```
.grid {
  display: grid;
  grid-row-gap: 8px;
  grid-column-gap: 8px;
  grid-template-columns: 1fr 2fr;
  grid-auto-flow: row;
  grid-auto-rows: 200px;
}

.grid_item {
  background-color: salmon;
}

.grid_item:last-child {
  background: red;
}

.grid_item {
  padding: 20px;
  font-size: 20px;
  font-family: sans-serif;
}
```

JS

```
Last saved less than a minute ago
```

GRID SHORTHAND

bit.ly/grid_short

Shorthands

grid shorthand with named
lines

grid-template-areas

grid-template-rows /

grid-template-columns,

grid-auto-flow,

grid-auto-rows /

grid-auto-columns

https://bit.ly/grid_short2

```
.grid {  
  grid-template-rows: [first] 200px [second] 100px [last];  
  grid-template-columns: [first] 1fr [second] 1fr [third] 1fr [last];  
  grid-template-areas: "header header header"  
                      "..... footer footer";  
}  
  
.grid {  
  grid:  
    [first] "header header header" 200px  
    [second] "..... footer footer" 100px [last] /  
    [first] 1fr [second] 1fr [third] 1fr [last];  
}
```

A screenshot of a CodePen editor window displaying a grid layout example. The grid consists of five items labeled 1 through 5. Item 1 is in the top-left position, item 2 is above it, item 3 is to its right, item 4 is below item 1, and item 5 spans two columns below item 1. The background colors are salmon for items 1, 2, and 3, and red for items 4 and 5.

The CodePen interface includes tabs for HTML, CSS, and JS, with the CSS tab currently active. The CSS code uses grid shorthand with named lines:

```
.grid {
  display: grid;
  grid-gap: 8px;
}

.grid {
  [first] "header header header" 200px
  [second] ".... footer footer" 100px [last] /
  [first] ifr [second] ifr [third] ifr [last];
}

/* Equivalent to: */
grid-template-rows: [first] 200px [second] 100px [last];
grid-template-columns: [first] 1fr [second] 1fr [third] 1fr [last];
grid-template-areas: "header header header"
  ".... footer footer";
}

.grid.item {
  background-color: salmon;
}

.grid_item:nth-child(1) {
  grid-column-start: second;
}

.grid_item:nth-child(2) {
  grid-column-start: first;
  grid-row-start: first;
}

.grid_item:last-child {
  background: red;
  grid-area: footer;
}
```

GRID SHORTHAND WITH NAMED LINES

bit.ly/grid_short2

Shorthands

grid shorthand

Imitating size in Post-CSS.

http://bit.ly/grid_short3

```
.grid {  
  display: grid;  
  grid: 400px / 500px;  
}
```

A screenshot of a CodePen editor window titled "grid shorthand 3". The preview area shows a single red grid item containing a heading and some lorem ipsum text. The editor interface includes tabs for HTML, CSS, JS, and a bottom navigation bar with "Console", "Assets", "Commands", "Delete", "Share", "Export", "Embed", and "Collections".

HTML

```
<div class="grid">
  <div class="grid-item">
    <h2>Heading</h2>
    <p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Omnis sunt expedita labore dicta quia nostrum magni magnam perferendis quam accusamus cupiditate unde ullam facilis quisquam, soluta accusantium & sit nihil?</p>
  </div>
</div>
```

CSS

```
.grid {
  display: grid;
  grid: 400px / 500px;
}

.grid_item {
  background-color: salmon;
  padding: 20px;
  /*align-self: end;*/}
```

GRID SHORTHAND

bit.ly/grid_short3

Shorthands

grid shorthand

Resetting grid properties.

```
.grid {  
    display: grid;  
    grid-template-rows: 200px 100px;  
    grid-template-columns: 1fr 1fr 1fr;  
    grid-template-areas: "header header header"  
                        "..... footer footer";  
    grid-auto-flow: column;  
    grid: none;  
}
```

CSS INTRINSIC AND EXTRINSIC SIZING

min-content, max-content, fit-content

CSS Intrinsic And Extrinsic Sizing

- The module defines additional ways to control size.
- Keywords that represent content-based "intrinsic" sizes and context-based "extrinsic" sizes.
- **min-content, max-content, fit-content.**
- **min-content and max-content** can be used for any of the properties that normally take a length (**width, height, min-width, flex-basis,...**).

Shorthands

min-content

The item becomes as large as it needs to be with the content becoming as small in the inline direction as possible.

```
.item {  
  width: -moz-min-content;  
  width: min-content;  
}
```

The screenshot shows a CodePen editor window titled "min-content". The page content is as follows:

min-content

Use `width: min-content` on the div, and the div now becomes only as large as it needs to be with the content becoming as small in the inline direction as possible. With a string of text this means that the text takes all of the soft-wrapping opportunities it can.

<https://www.smashingmagazine.com/2018/01/understanding-sizing-css-layout/>

Lorem ipsum dolor sit amet consectetur adipisicing elit.

The CSS code in the editor is:

```
border-color: #17242D;
padding: 10px;
font-size: 2rem;
margin: 2rem;
}

.min {
  width: -moz-min-content;
  width: min-content;
  border-color: #D73B38;
}

div {
  border: 10px solid;
  border-radius: 4px;
```

MIN-CONTENT

bit.ly/grid_min

Shorthands

max-content

The item becomes large enough to contain the content.

```
.item {  
    width: -moz-max-content;  
    width: max-content;  
}
```

The screenshot shows a CodePen editor window titled "max-content". The URL in the address bar is <https://codepen.io/matuzo/pen/wywMOg?editors=1100>. The title of the codepen is "max-content" by Manuel Matuzovic PRO.

The main content area displays the heading "max-content" and a paragraph explaining the behavior of the `max-content` property:

The box becomes larger enough to contain the content if it gets as larger in the inline dimension as possible. Our string of text now stretches out and does no wrapping at all. This will cause overflows should it become wider than the available width this div has to grow into.

A link to <https://www.smashingmagazine.com/2018/01/understanding-sizing-css-layout/> is present.

The visual output shows a red-bordered box containing the text "Lorem ipsum dolor sit amet consectetur adipisicing elit.".

The right side of the interface contains the code editor with three tabs: HTML, CSS, and JS. The HTML tab shows the markup, the CSS tab shows the styles applied, and the JS tab is currently empty.

```
HTML
1 <h1>max-content</h1>
2
3 <blockquote>
4   The box becomes larger enough to
      contain the content if it gets as larger
      in the inline dimension as possible. Our
      string of text now stretches out and
      does no wrapping at all. This will cause
      overflows should it become wider than
      the available width this div has to grow
      into.
5 </blockquote>
6 <cite><a href="https://www.smashingmagazine.com/2018/01/understanding-sizing-css-layout/">https://www.smashingmagazine.com/2018/01/understanding-sizing-css-layout/</a></cite>
7

CSS
1 .wrapper {
2   border-color: #17242D;
3   padding: 10px;
4   font-size: 2rem;
5   margin: 2rem;
6 }
7
8 .min {
9   width: -moz-max-content;
10  width: max-content;
11  border-color: #D73B38;
12 }
13
14 div {
15   border: 10px solid #D73B38;
16 }

JS
```

MAX-CONTENT

bit.ly/grid_max

The screenshot shows a CodePen editor window titled "min-content grid layout". The URL is <https://codepen.io/matuzo/pen/GQKZqN?editors=1100>. The preview area displays a grid of six red boxes labeled Alfa, Bravo, Charlie, Delta, Echo, and Foxtrot. The "Bravo" box contains the text "has some additional content which will wrap". The "CSS" panel contains the following code:

```
.grid {  
  border: 2px solid #17242D;  
  width: 500px;  
  display: grid;  
  grid-gap: 20px;  
  grid-template-columns: min-content min-content min-content;  
}
```

MIN-CONTENT FOR TRACK SIZING

bit.ly/grid_min2

The screenshot shows a CodePen editor window titled "max-content grid layout". The preview area displays a grid of six red rounded rectangular boxes labeled Alfa, Bravo, Charlie, Delta, Echo, and Foxtrot. The "Bravo" box contains the text "Bravo has some additional content which will wrap". The "Foxtrot" box has a vertical line pointing down to it from the "Charlie" box. The CodePen interface includes tabs for "HTML", "CSS (SCSS)", and "JS", each containing the following code:

```
HTML
<div class="grid">
  <div class="item">Alfa</div>
  <div class="item">Bravo has some additional content which will wrap</div>
  <div class="item">Charlie</div>
  <div class="item">Delta</div>
  <div class="item">Echo</div>
  <div class="item">Foxtrot</div>
</div>
```

```
CSS (SCSS)
.grid {
  border: 2px solid #17242D;
  width: 500px;
  display: grid;
  grid-gap: 20px;
  grid-template-columns: 1fr 1fr 1fr;
  // grid-template-columns: auto auto auto;
  grid-template-columns: max-content max-content max-content;
}
```

```
JS
```

MAX-CONTENT FOR TRACK SIZING

bit.ly/grid_max2

Shorthands

fit-content()

fit-content() takes a length or percentage as a value.

When you use **fit-content** for track sizing, the track will act like **max-content** until it gets to the size of the passed in value.

```
.grid {  
  display: grid;  
  grid-template-columns: fit-content(10rem) fit-  
content(10rem) fit-content(10rem);  
}
```

A screenshot of a CodePen editor window titled "fit-content". The URL is <https://codepen.io/matuzo/pen/MQgywB?editors=1100>. The preview area shows a grid of six red boxes with white text: "Alfa", "Bravo has some additional content which will wrap", "Charlie" in the top row, and "Delta", "Echo", "Foxtrot" in the bottom row. The "HTML" panel contains the following code:

```
<div class="grid">
  <div class="item">Alfa</div>
  <div class="item">Bravo has some additional content which will wrap</div>
  <div class="item">Charlie</div>
  <div class="item">Delta</div>
  <div class="item">Echo</div>
  <div class="item">Foxtrot</div>
</div>
```

The "CSS (SCSS)" panel contains the following SCSS code:

```
.grid {
  border: 2px solid #17242D;
  width: 500px;
  display: grid;
  grid-gap: 20px;
  // grid-template-columns: min-content min-content min-content;
  // grid-template-columns: max-content max-content max-content;
  grid-template-columns: fit-content(10rem) fit-content(10rem) fit-content(10rem);
}
```

The "JS" panel is empty.

FIT-CONTENT

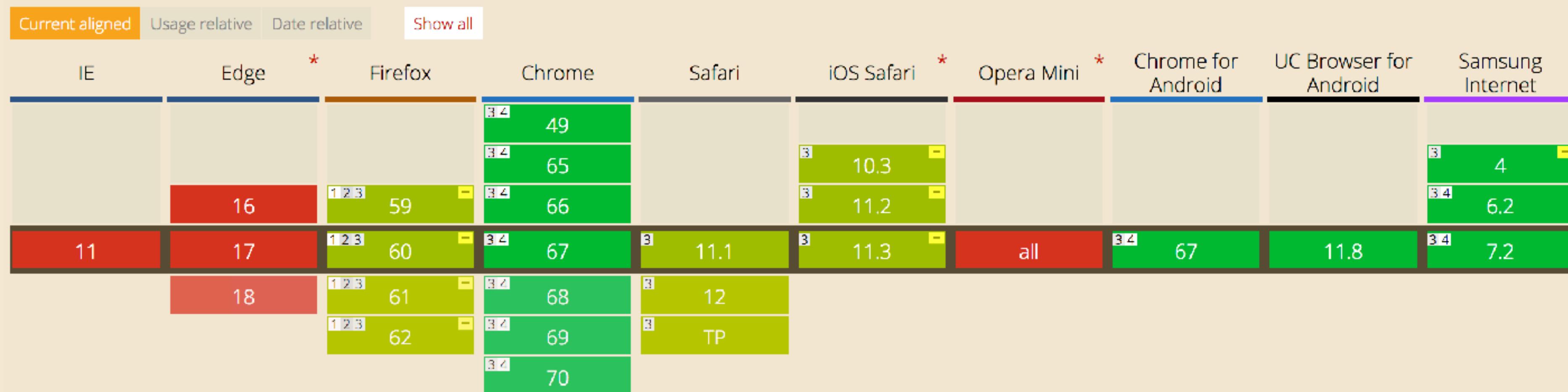
bit.ly/grid_fit

Browsersupport

Intrinsic & Extrinsic Sizing - WD

Allows for the heights and widths to be specified in intrinsic values using the `max-content`, `min-content`, `fit-content` and `stretch` (formerly `fill`) properties.

Usage
Global: 72.04% + 17.27% = 89.31%
unprefixed: 69.6% + 1.69% = 71.29%



PATTERNS

Some common patterns



EXERCISE 4

A screenshot of a web browser window titled "CodePen - CSS Grid Layout Site". The page has a dark blue header with the word "Header" in white. The main content area contains five identical paragraphs of placeholder text ("Lorem ipsum dolor sit amet consectetur, adipisciing elit. Eigendi delectus officia impedit ipsa, enim sequi sint aliquam suscipit molestiae eos perferendis consequuntur nam, consequatur ipsam similiq..."). Below the content is a dark blue footer bar with the text "© 2017". The browser interface includes a search bar, a toolbar with various icons, and a tab bar at the top.

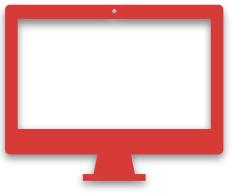


EXERCISE 4

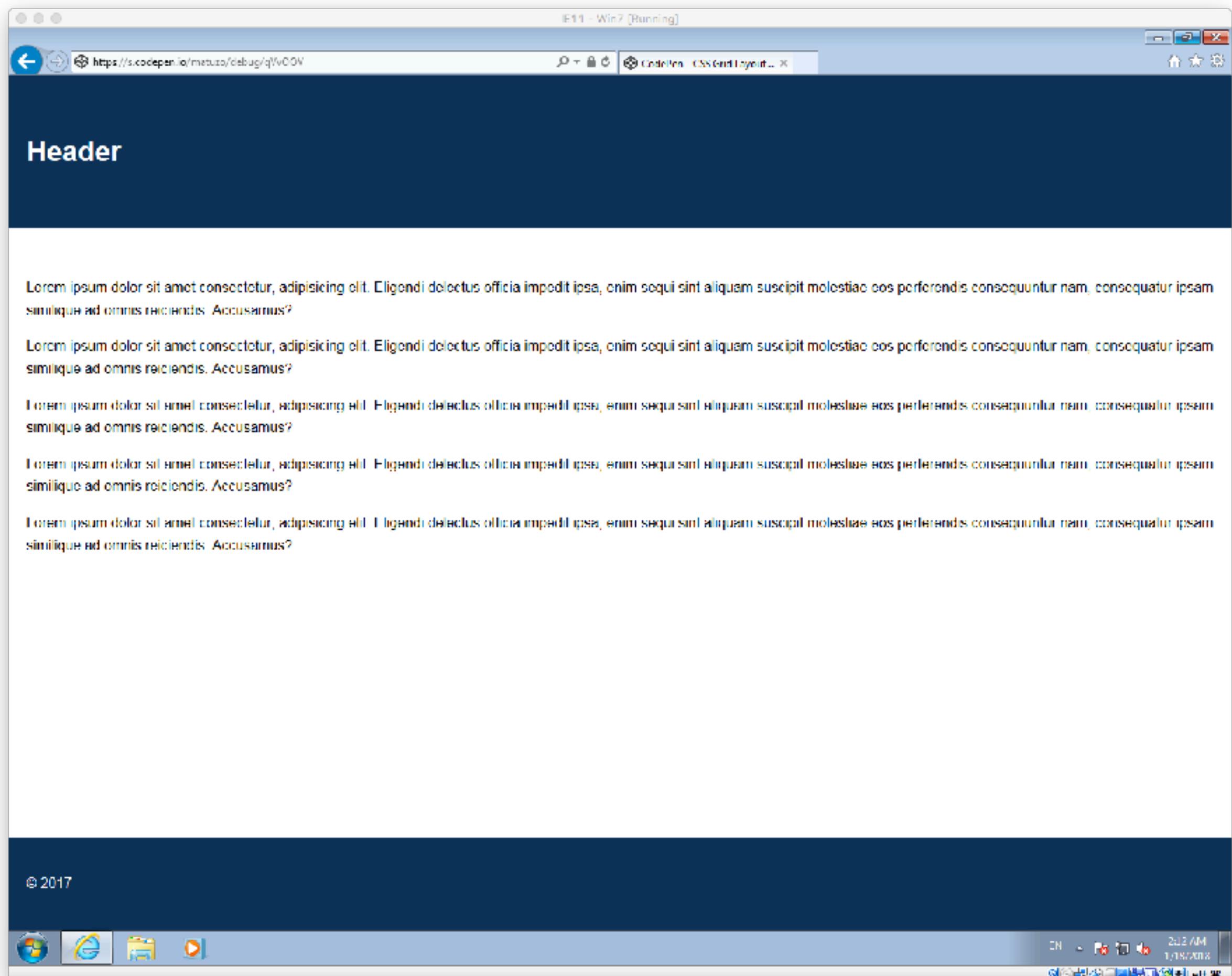
Sticky footer

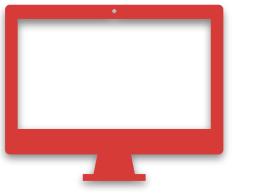
- Try to implement a sticky footer pattern with grid.
- Footer Sticky at the bottom of the page.
- Footer moves with the content if the content is longer than the page.
- Flexible height for header and footer.

https://bit.ly/grid_sticky

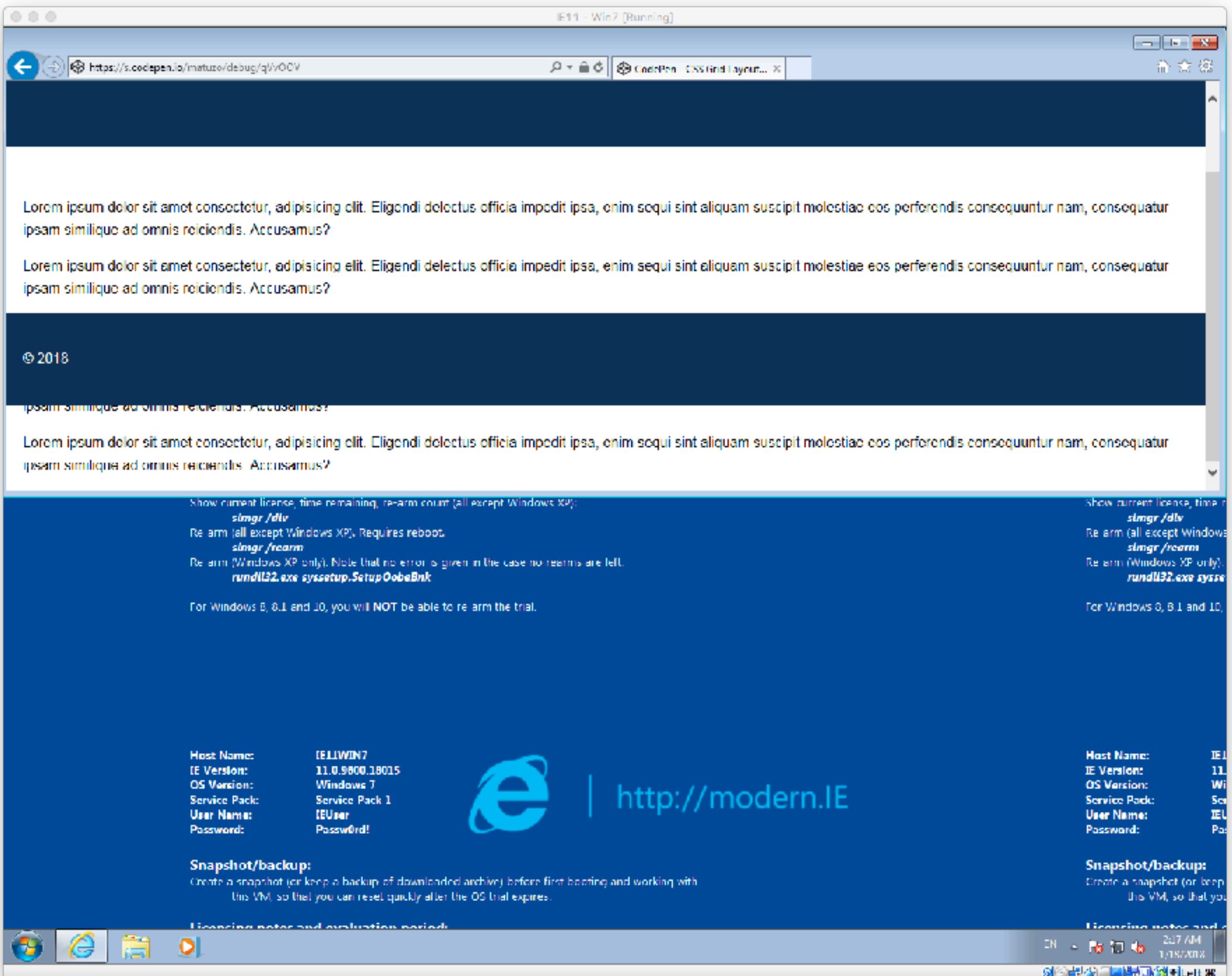


EXERCISE 4.5





EXERCISE 4.5

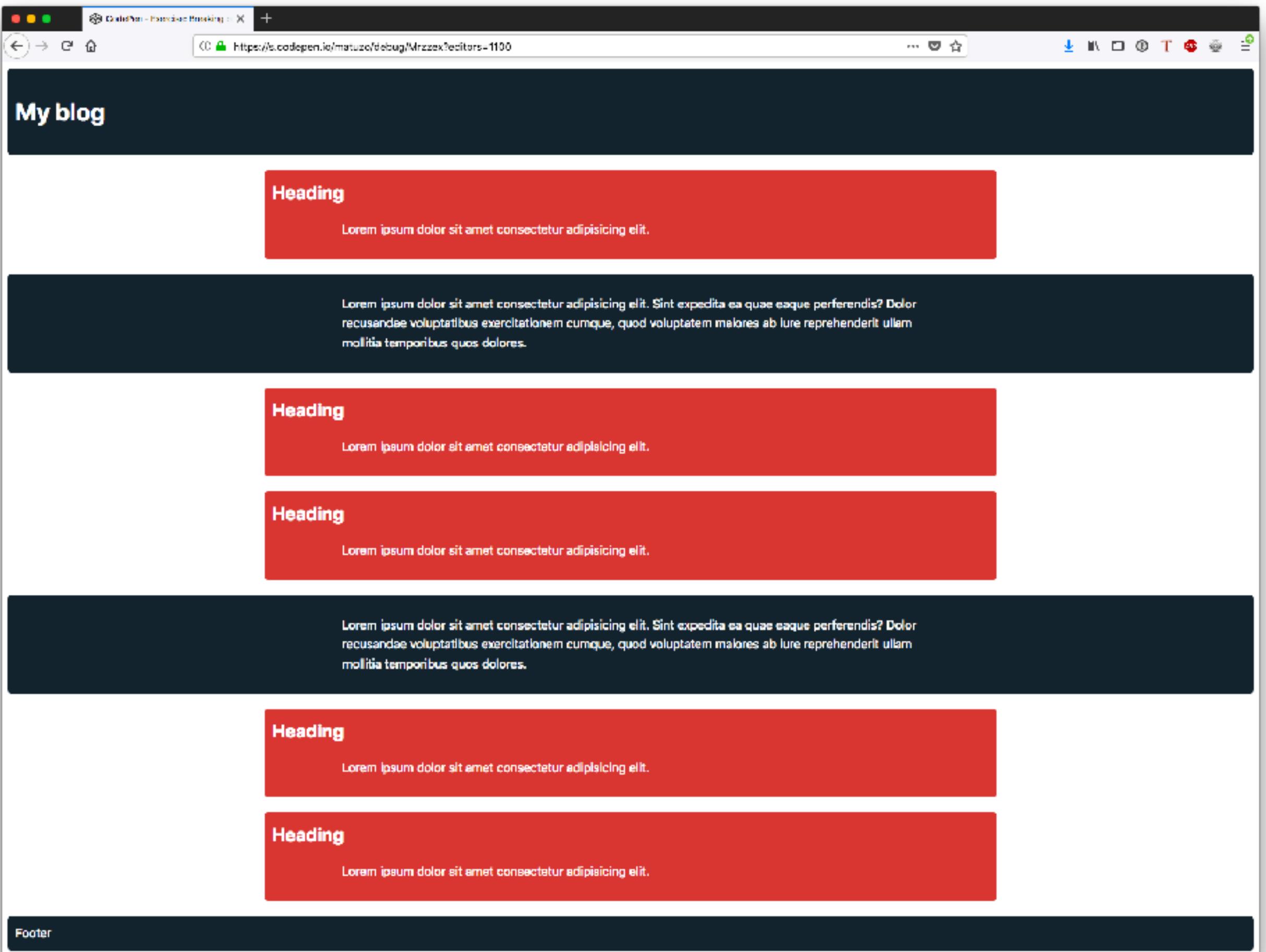


STICKY FOOTER IN IE: 😞

https://bit.ly/grid_sticky_ie



EXERCISE 5



ACCESSIBILITY

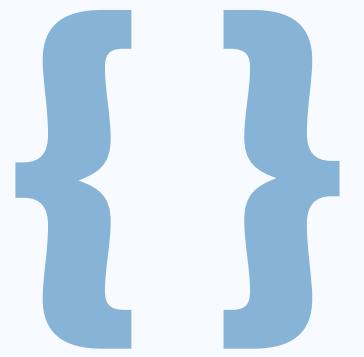
„Authors must use order and the grid-placement properties only for visual, not logical, reordering of content. Style sheets that use these features to perform logical reordering are non-conforming.”

– <https://www.w3.org/TR/css-grid-1/#order-accessibility>

{} {}}



```
<ul>
  <li>
    <a href="#">
      
    </a>
  </li>
  ...
</ul>
```



```
ul {  
  display: grid;  
  grid-template-columns: repeat(auto-fill, minmax(180px, 1fr));  
  grid-gap: 20px;  
  grid-auto-rows: 180px;  
}
```

A grid with as many 180px wide columns as possible



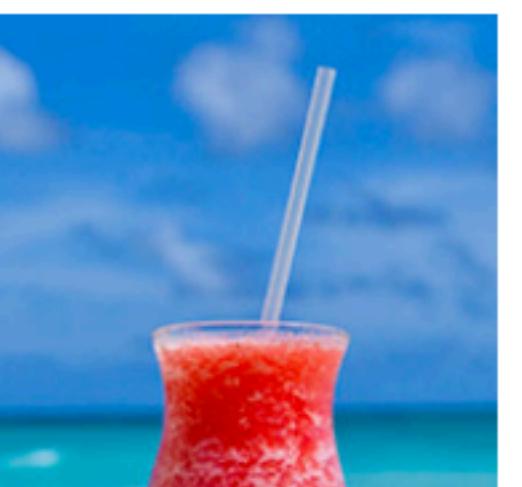
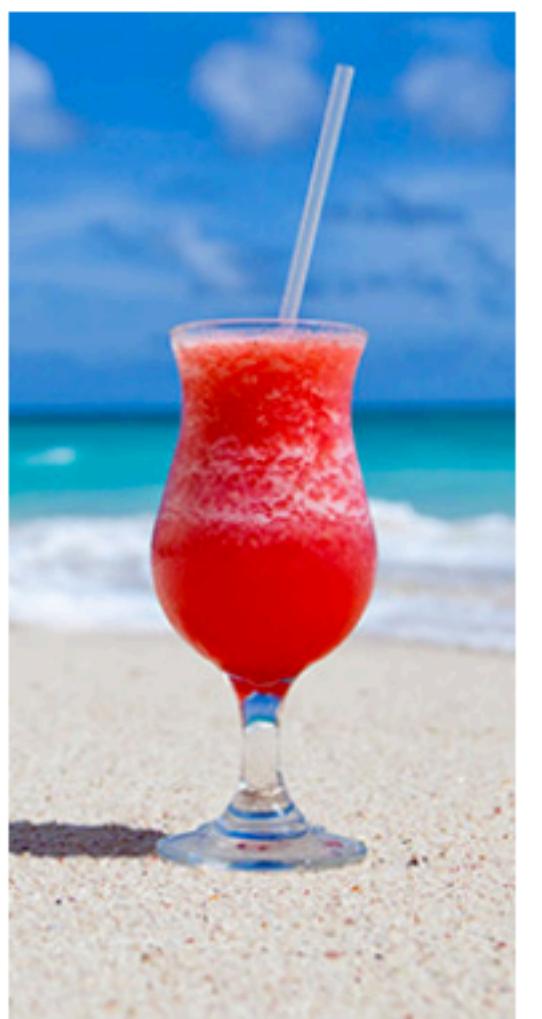
```
.long {  
  grid-row: span 2;  
}
```

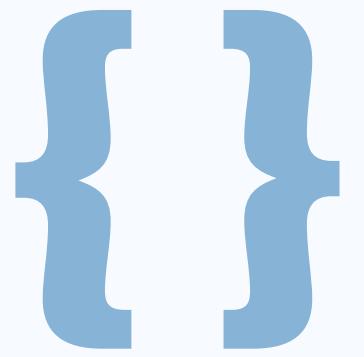
{ }

```
.large {  
  grid-row: span 2;  
}
```

```
.large1 {  
  grid-column: span 2 / -1;  
}
```

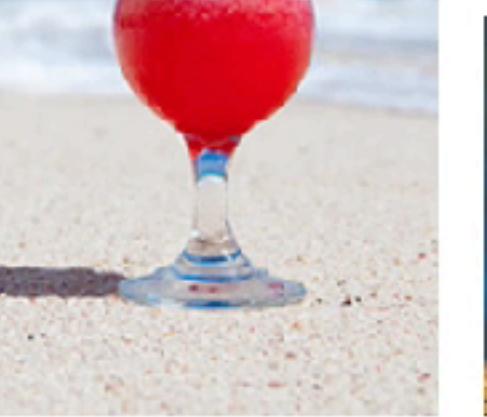
```
.large2 {  
  grid-column: 1 / span 2;  
}
```





```
ul {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(180px, 1fr));
  grid-gap: 20px;
  grid-auto-rows: 180px;
  grid-auto-flow: dense;
}
```

grid-auto-flow tries to fill the gaps in a grid





Accessibility

- Planing and design start with the content and source order.
- If something needs to be moved significantly, move it in the DOM and not visually.
- Don't be tempted to compromise on semantics and flatten the structure due to design or developer experience reasons .

Development process

- Start with a semantic and well structured document.
- Create the grid and check if the structure still makes sense.
- Test your pages and components with the keyboard.
- If required, change the source order.

Links

- <http://adrianroselli.com/2015/09/source-order-matters.html>
- <http://adrianroselli.com/2015/10/html-source-order-vs-css-display-order.html>
- https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout/CSS_Grid_Layout_and_Accessibility
- <https://www.w3.org/TR/css-grid-1/#order-accessibility>

STACKING ORDER

Using z-index with grid items

Stacking order

- The rendering order of elements on the z-axis can be influenced by changing the `z-index` property.
- This applies only to items which form a *stacking context*.
- A stacking context is formed, anywhere in the document, by any element in the following scenarios:

Stacking context

- position **with a value of relative, absolute, fixed or sticky.**
- opacity **less than 1.**
- mix-blend-mode **value other than normal.**
- Elements **with any of the following properties with values other than none:** transform, filter, perspective, clip-path, mask / mask-image / mask-border
- Element **with an isolation value isolate.**

Stacking context

- Elements with a `-webkit-overflow-scrolling` value touch.
- Elements with a `will-change` value specifying any property that would create a stacking context on non-initial value.
- Flex items.
- Grid items.

The screenshot shows a web-based code editor for a CSS stacking order demonstration. On the left, there's a visual representation of a stack of colored boxes labeled 1 through 15. To the right of the boxes is a large red button labeled "ON/OFF". Below the button is a code editor interface with three tabs: "HTML", "CSS (less)", and "JS".

HTML:

```
<button>ON/OFF</button>


<div class="parent">
    <div class="box red">1</div>
  </div>
  <div class="parent">
    <div class="box green">2</div>
  </div>
  <div class="parent">
    <div class="box blue">3</div>
  </div>
  <div class="parent">
    <div class="box red">4</div>
  </div>
  <div class="parent">
    <div class="box green">5</div>
  </div>
  <div>
    <div>
      <div>9</div>
      <div>11</div>
      <div>12</div>
      <div>13</div>
      <div>14</div>
      <div>15</div>
    </div>
  </div>


```

CSS (less):

```
.on {
  .parent:nth-child(1) {
    opacity: 0.9;
  }
  .parent:nth-child(3) {
    transform: rotate(0deg);
  }
  .parent:nth-child(5) {
    perspective: 1000px;
  }
  .parent:nth-child(7) {
    will-change: transform;
  }
  .parent:nth-child(9) {
    filter: grayscale(100%);
  }
  .parent:nth-child(11) {
    mix-blend-mode: exclusion;
  }
  .parent:nth-child(13) {
    isolation: isolate;
  }
}
```

JS:

```
;
```

At the bottom of the editor, there are tabs for "Console", "Assets", "Commands", and "X". On the far right, there are buttons for "Save", "Fork", "Settings", "Change View", and user profile information.

STACKING ORDER

bit.ly/grid_stacking

Stacking order

It's possible to stack grid items.

Their positions on the z-axis is determined by the **z-index** property.

```
.grid-item {  
    grid-column-start: 1;  
}
```

```
.grid-item2 {  
    grid-column-start: 1;  
    z-index: 1;  
}
```

CSS grid stacking order

A PEN BY Manuel Matuzovic PRO

Save Fork Settings Change View

* HTML

```
<div class="grid">
  <div class="item">1</div>
  <div class="item">2</div>
  <div class="item">3</div>
  <div class="item">4</div>
</div>
```

* CSS

```
.grid {
  display: grid;
  grid-template-columns: 100px 100px;
  grid-template-rows: 100px 100px;
  //grid-auto-flow: column;
}

@for $i from 1 through 4 {
  .item:nth-child(#{$i}) {
    background-color: hsl(90 * $i, 100%, 50%);
  }
}

.item:nth-child(1) {
  //z-index: 1;
}

.item:nth-child(2) {
  transform: translateY(-20px) translateX(-20px);
}
```

* JS

```
1
```

The screenshot shows a code editor interface with three panels: HTML, CSS, and JS. The HTML panel contains a simple grid structure with four items. The CSS panel contains styling for the grid, including a color gradient for each item based on its index and a transform for item 4. The JS panel is empty. Below the editor is a preview area showing a 2x2 grid of four colored squares (green, cyan, purple, red) with square 4 shifted down and left relative to square 3.

STACKING ORDER GRID

bit.ly/grid_stacking1

RESPONSIVE WEBDESIGN



EXERCISE 6

My blog

Posts

Lorem ipsum dolor sit amet consectetur adipisicing elit. Reprehenderit quam praesentium?

Post

Lorem ipsum dolor sit amet consectetur adipisicing elit.

Post 2

Lorem ipsum dolor sit amet consectetur adipisicing elit.

Post 3

Lorem ipsum dolor sit amet consectetur adipisicing elit.

Latest articles

- Post 1
- Post 2
- Post 3

Latest comments

- Post 1
- Post 2
- Post 3

Footer

My blog

Posts

Lorem ipsum dolor sit amet consectetur adipisicing elit. Reprehenderit quam praesentium?

Post

Lorem ipsum dolor sit amet consectetur adipisicing elit.

Post 2

Lorem ipsum dolor sit amet consectetur adipisicing elit.

Post 3

Lorem ipsum dolor sit amet consectetur adipisicing elit.

Latest articles

- Post 1
- Post 2
- Post 3

Latest comments

- Post 1
- Post 2
- Post 3

Footer

My blog

Posts

Lorem ipsum dolor sit amet consectetur adipisicing elit. Reprehenderit quam praesentium?

Post

Lorem ipsum dolor sit amet consectetur adipisicing elit.

Post 2

Lorem ipsum dolor sit amet consectetur adipisicing elit.

Post 3

Lorem ipsum dolor sit amet consectetur adipisicing elit.

Latest articles

- Post 1
- Post 2
- Post 3

Latest comments

- Post 1
- Post 2
- Post 3

Footer

The screenshot shows a CodePen editor interface with the following details:

- Title:** Basic grid system functionality
- Author:** Manuel Matuzovic
- Preview:** A grid of 12 red boxes labeled 1 through 12, demonstrating various CSS Grid properties like span, repeat, and column-end.
- HTML:**

```
60 <div class="grid">
61   <div class="item item4">
62     <h2>1</h2>
63   </div>
64   <div class="item item4">
65     <h2>2</h2>
66   </div>
67   <div class="item item4">
68     <h2>3</h2>
69   </div>
70
71   <div class="item item6">
72     <h2>1</h2>
73   </div>
74   <div class="item item6">
75     <h2>2</h2>
76   </div>
77
78   <div class="item item8">
79     <h2>1</h2>
80   </div>
81   <div class="item item4">
82     <h2>2</h2>
83   </div>
84   <div class="item item3">
85     <h2>1</h2>
86   </div>
87   <div class="item item3">
88     <h2>2</h2>
89   </div>
90   <div class="item item3">
91     <h2>3</h2>
92   </div>
93   <div class="item item3">
```
- CSS (SCSS):**

```
1 .grid {
2   display: grid;
3   grid-template-columns: repeat(12, 1fr);
4   grid-gap: 20px;
5 }
6
7 @for $i from 1 through 12 {
8   .item#{$i} {
9     grid-column-end: span $i;
10 }
11 }
```
- JS:** None
- Footer:** Last saved 9 months ago, Delete, Share, Export, Embed, Collections

BASIC GRID SYSTEM FUNCTIONALITY

bit.ly/grid_mq3

The screenshot shows a CSS Grid layout editor interface. On the left, a preview window displays a layout with 6 red rectangular items labeled 1 through 6. Item 1 is in the top row, spanning two columns. Item 2 is in the second column of the top row. Item 3 is in the third column of the top row. Item 4 is in the first column of the bottom row. Item 5 is in the second column of the bottom row. Item 6 is in the third column of the bottom row. To the right of the preview are four code editors: HTML, SCSS, CSS, and JS. The HTML editor shows the structure of the grid. The SCSS editor contains the following code:

```
1 $site_max_width: 1200px;
2 $gap: 20px;
3 $max_columns: 4;
4
5 // $win-width: ($site_max_width - ($gap * ($max_columns - 1))) / $max_columns;
6
7 .grid {
8   display: grid;
9   grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
10  grid-gap: $gap;
11  max-width: $site_max_width;
12 }
13
14 * {
15   box-sizing: border-box;
16 }
```

The CSS editor contains the following code:

```
1 site_max_width: 1200px;
2 gap: 20px;
3 max_columns: 4;
4
5 // win-width: ($site_max_width - (gap * (max_columns - 1))) / max_columns;
6
7 .grid {
8   display: grid;
9   grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
10  grid-gap: gap;
11  max-width: site_max_width;
12 }
13
14 * {
15   box-sizing: border-box;
16 }
```

The JS editor is empty. At the bottom of the interface, there are tabs for Console, Assets, Comments, and a file menu with options like Delete, Share, Export, Embed, and Collections.

RWD WITHOUT MEDIA QUERIES

bit.ly/grid_mq4

PROGRESSIVE ENHANCEMENT

Progressive Enhancement

- Progressive enhancement is a philosophy.
- One principle: fault tolerance.
- Browsers must ignore anything they don't understand in HTML and CSS.
- Fault tolerance is the reason progressive enhancement works.
- The core focus is the content and everything builds upon that.



<https://adaptivewebdesign.info/1st-edition/read/chapter-1.html>

*"Progressive enhancement ensures that all content
(that is to say the information contained in a website)
is both available to and usable by anyone, regardless of
her location, the device she is using to access that
information, or the capabilities of the program she is
using to access that content"*

– Aaron Gustafson

Progressive Enhancement

- PE doesn't require that you provide the same experience in different browsers.
- PE doesn't preclude you from using the latest and greatest technologies.
- PE just asks you to apply technologies in an intelligent way, layer-upon-layer.
- Progressive enhancement = excellent customer service.

Progressive Enhancement in HTML

Due to HTML's fault tolerance this markup will work even in very old (e.g. IE 8) or basic (e.g. Lynx) browsers.

```
<header>
  <h1>My site</h1>
  <nav>
    <ol>
      <li><a href="#links">Links</a></li>
    </ol>
  </nav>
</header>
```

Progressive Enhancement in HTML

Using the **picture** element to provide modern browsers with **webp** images.

Browsers that either don't understand **webp** or the **picture** element fall back to the **img**.

```
<picture>
  <source srcset="opera.webp" type="image/webp">
    
</picture>
```

Progressive Enhancement in HTML

Instead of seeing nothing when an image fails to load, users will at least know what's in the picture ("The Oslo Opera House").

```

```

Progressive Enhancement in HTML

Using the **title** attribute on an **abbr** tag enhances the experience for both blind and sighted users.

```
<abbr title="World Wide Web Consortium">W3C</abbr>
```

Progressive Enhancement in CSS

A browser reads each rule set and examines it. If it encounters something it doesn't understand, it experiences a parsing error.

```
p {  
    color: red;  
    font-weight: bold;  
}
```

Progressive Enhancement in CSS

Employ parsing errors to advantage by providing fallback declarations.

```
div {  
    width: 50vw;  
    width: 50vmax;  
}
```

Progressive Enhancement in CSS

By preferring (min-width) over max-width in media queries, not only smartphone users but users of browsers than can't interpret media queries will get at least some kind of experiences.

```
div {  
    width: 90%;  
}  
  
@media (min-width: 48em) {  
    div {  
        display: flex;  
    }  
}
```

Graceful Degradation



Progressive Enhancement



Progressive Enhancement in JS

Only capable browsers will cache contents offline and improve performance and user experience.

```
if ('serviceWorker' in navigator) {
  navigator.serviceWorker.register('sw.js').then(function(reg) {
    // ...
  }).catch(function(error) {
    // ...
  });
};
```

Progressive Enhancement

- **float** and **clear** have no effect
- **vertical-align** has no effect
- **flex** items become grid items
- **block**, **inline-block** or **table-cell** items become grid items.
- **column-** properties have no effect

Progressive Enhancement

It's not necessary to create multiple layouts.

```
.grid {  
  display: flex;  
  flex-wrap: wrap;  
  display: grid;  
  grid-template-columns: repeat(5, 1fr);  
}
```

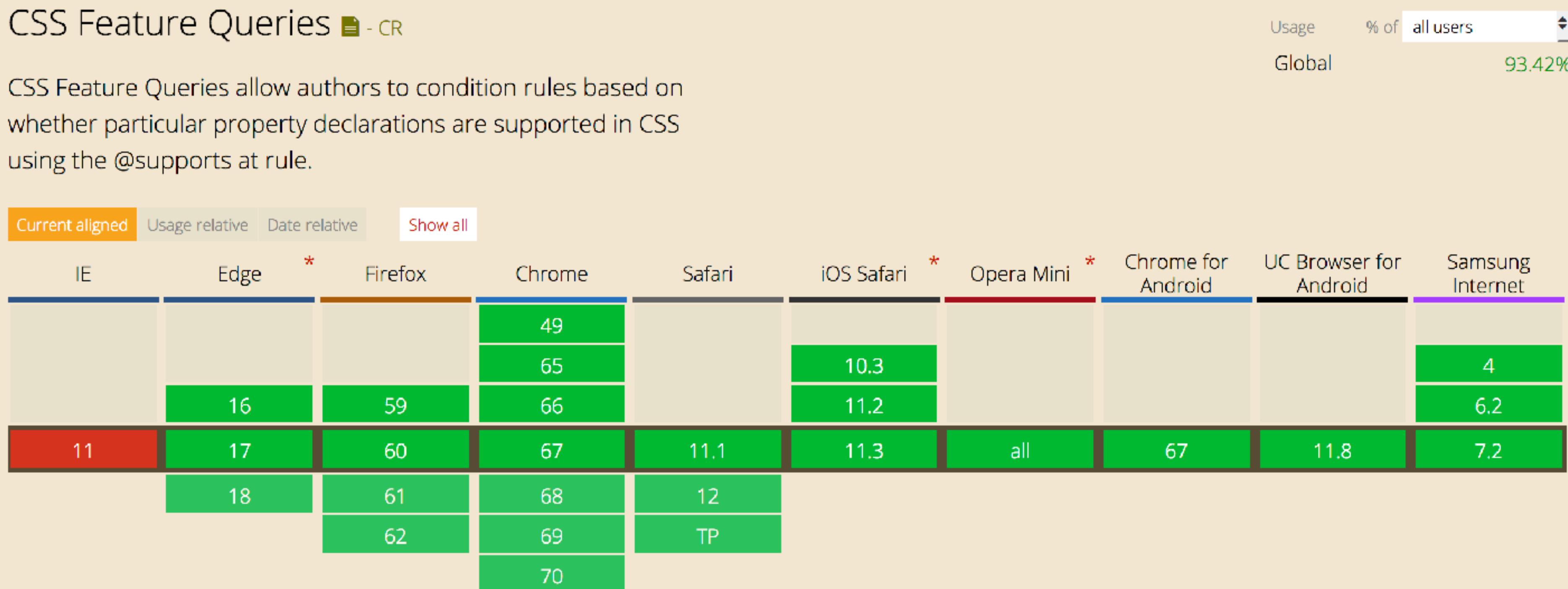
Progressive Enhancement

Feature Queries.

Check for property + value.

```
@supports (display: grid) {  
  .grid {  
    margin-right: 0;  
  }  
}
```

Feature Queries



<http://caniuse.com/#search=feature%20queries>

Detecting Support

- We don't need tools like Modernizr.
- All browsers that understand Grid understand Feature Queries as well.
- It might be necessary to differentiate between the old and the new spec.
- In newer versions of Edge queries for both the old and the new syntax evaluate to true.

Feature detection

Detecting the new spec

```
@supports(display: grid) {  
  div {  
    margin: 0;  
  }  
}
```

Detecting the old spec

```
@supports(display: -ms-grid) {  
  div {  
    margin: 0;  
  }  
}
```

The screenshot shows a browser-based code editor interface with the following details:

- Title Bar:** "display: grid support test" and "Secure https://codepen.io/matuzevic/pen/PKwewD".
- Toolbar:** Save, Fork, Settings, Change View.
- Code Area:** Contains four blocks of CSS code using the `@supports` pseudo-class to target different browser features.
- HTML Preview:** Shows a visual representation of the CSS output, consisting of four colored boxes (green, red, green, red) arranged in a 2x2 grid pattern.
- Code Sections:** HTML, CSS, and JS.
- Bottom Bar:** Includes tabs for Console, Assets, Commands, and a status message "Last saved less than a minute ago".

```
HTML
28      background-color: green;
29    }
30  
31  
```

```
CSS
23  @supports(display: grid) {
24    div:nth-child(1) {
25      background-color: green;
26    }
27
28  @supports(display: -ms-grid) {
29    div:nth-child(2) {
30      background-color: green;
31    }
32
33  @supports(grid-area: auto) {
34    div:nth-child(3) {
35      background-color: green;
36    }
37
38  @supports(display: -ms-grid) and (display: grid) {
39    div:nth-child(4) {
40      background-color: green;
41    }
42
43 }
```

GRID FEATURE TEST IN CHROME 66

bit.ly/grid_feature1

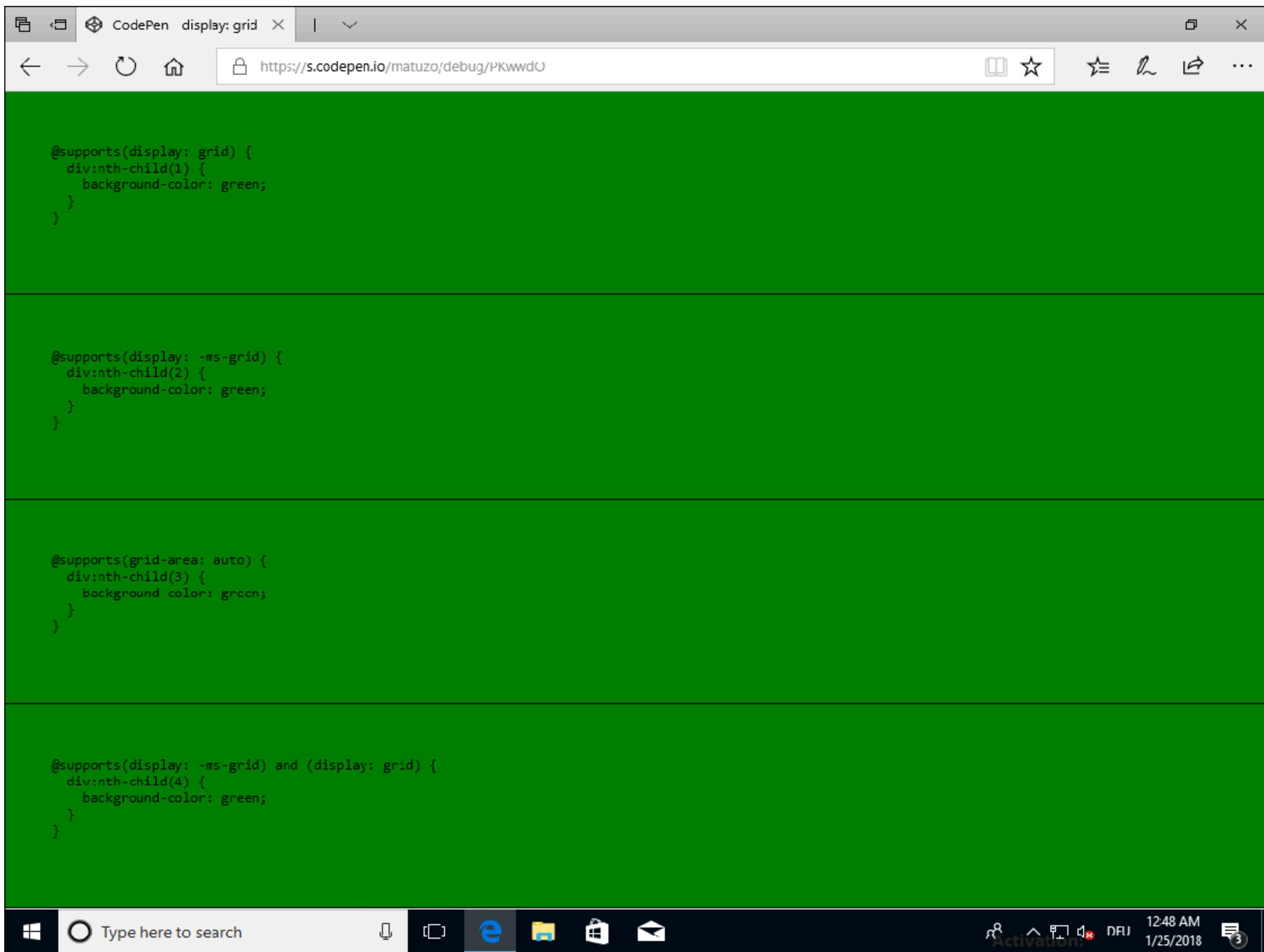
```
@supports(display: grid) {  
  div:nth-child(1) {  
    background-color: green;  
  }  
}
```

```
@supports(display: -ms-grid) {  
  div:nth-child(2) {  
    background-color: green;  
  }  
}
```

```
@supports(grid-area: auto) {  
  div:nth-child(3) {  
    background-color: green;  
  }  
}
```

```
@supports(display: -ms-grid) and (display: grid) {  
  div:nth-child(4) {  
    background-color: green;  
  }  
}
```





GRID FEATURE TEST IN EDGE 16

bit.ly/grid_feature1

Progressive Enhancement

Feature Queries.

Only change what's necessary.

```
.grid-item {  
    float: left;  
    margin-right: 20px;  
}  
  
@supports(display: flex) {  
    .grid-item {  
        margin-right: 0;  
    }  
}
```

Enhance!

Use Grid today!

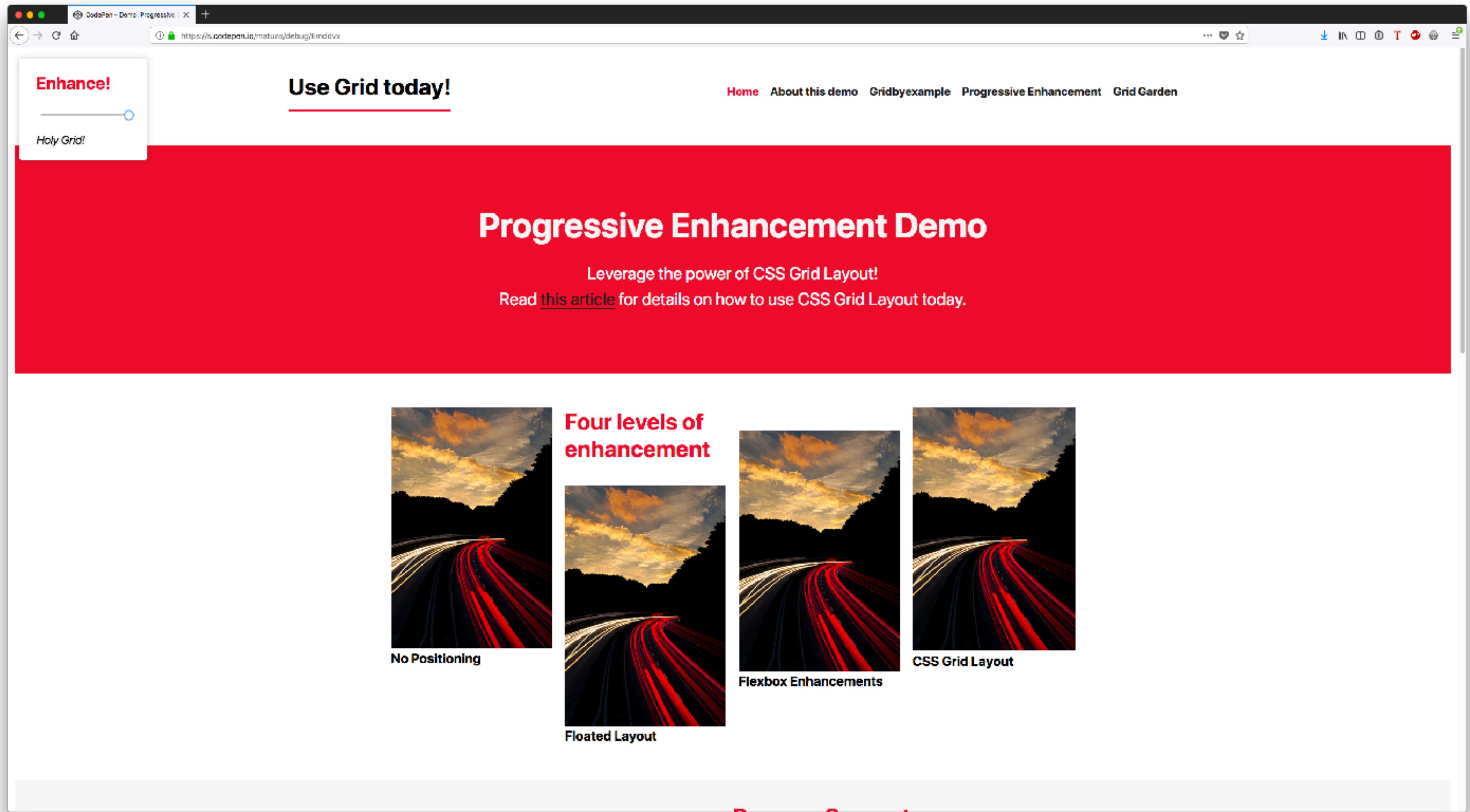
Home About this demo Gridbyexample Progressive Enhancement Grid Garden

Progressive Enhancement Demo

Leverage the power of CSS Grid Layout!
[Read this article](#) for details on how to use CSS Grid Layout today.

Four levels of enhancement

- No Positioning
- Floated Layout
- Flexbox Enhancements
- CSS Grid Layout



Progressive Enhancement Example

Markup: section, heading
and 4 articles

```
<section>
  <h2>Four levels of enhancement</h2>

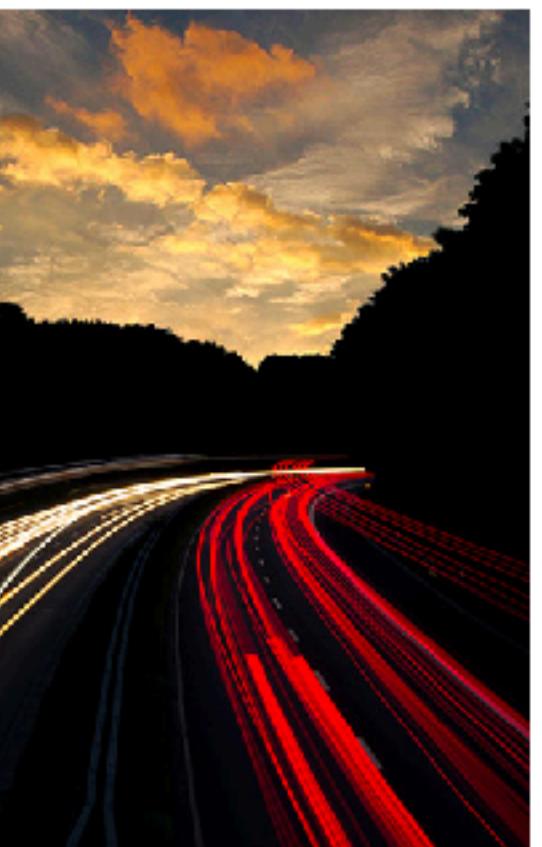
  <article>
    <h3>No Positioning</h3>
  </article>
  <article>
    ...
  </article>
</section>
```

Four levels of enhancement

No Positioning



Floated Layout



Flexbox Enhancements

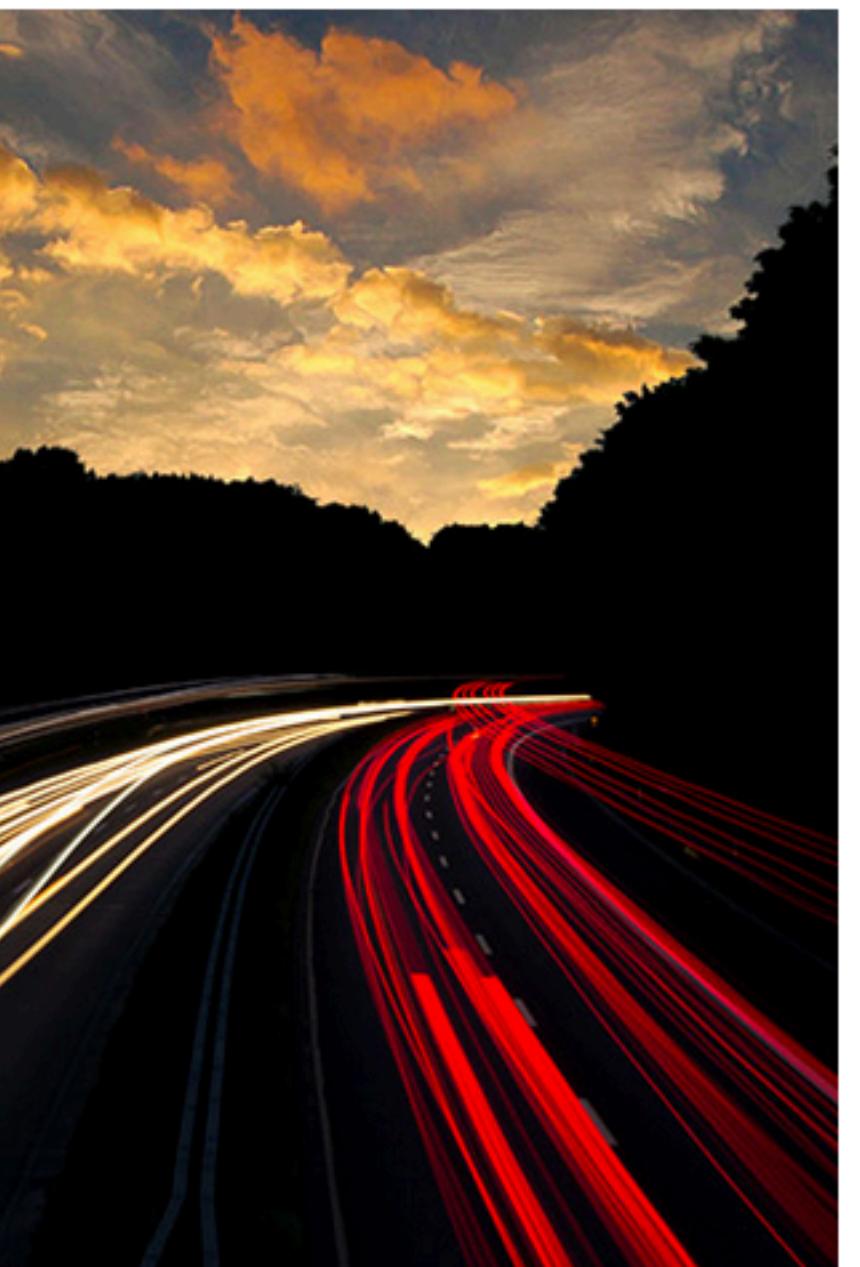
Progressive Enhancement Example

Floating articles

```
article {  
    float: left;  
    width: 24.25%;  
}  
  
article:not(:last-child) {  
    margin-right: 1%;  
}  
  
section:after {  
    clear: both;  
    content: "";  
    display: table;  
}
```

Four levels of enhancement

No Positioning



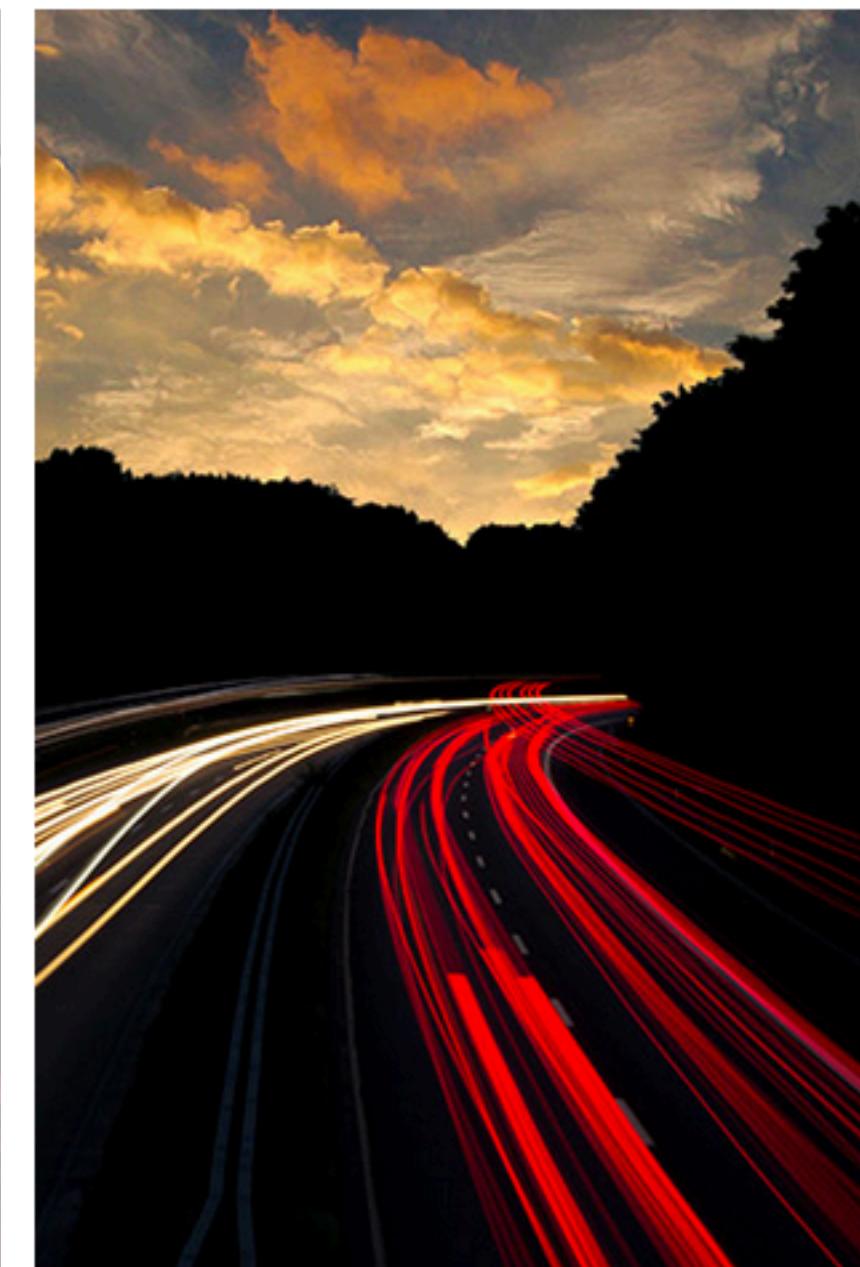
Floated Layout



Flexbox Enhancements



CSS Grid Layout



Progressive Enhancement Example

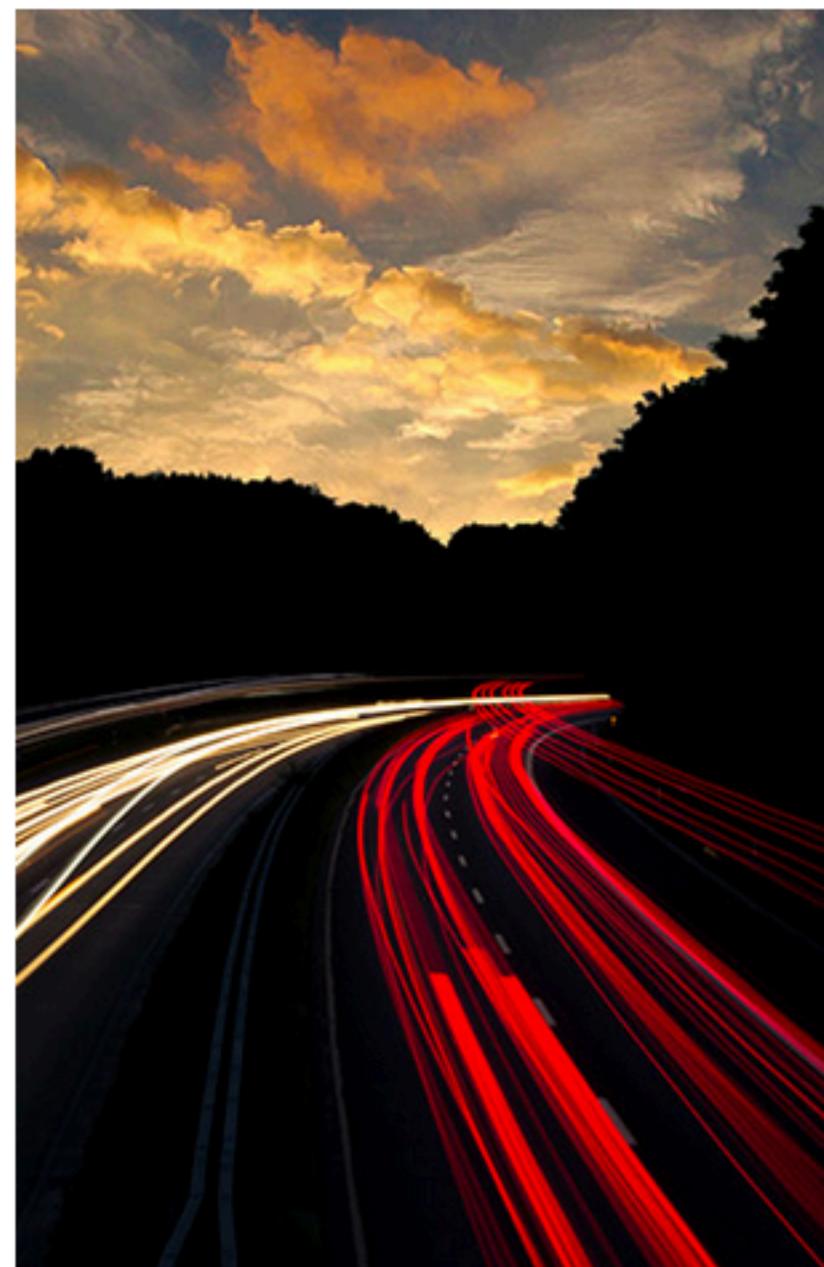
Enhancing with Flexbox.

```
article {  
    float: left;  
    width: 24.25%;  
  
    display: flex;  
    flex-direction: column;  
}  
  
h3 {  
    order: 1;  
}
```

Four levels of enhancement



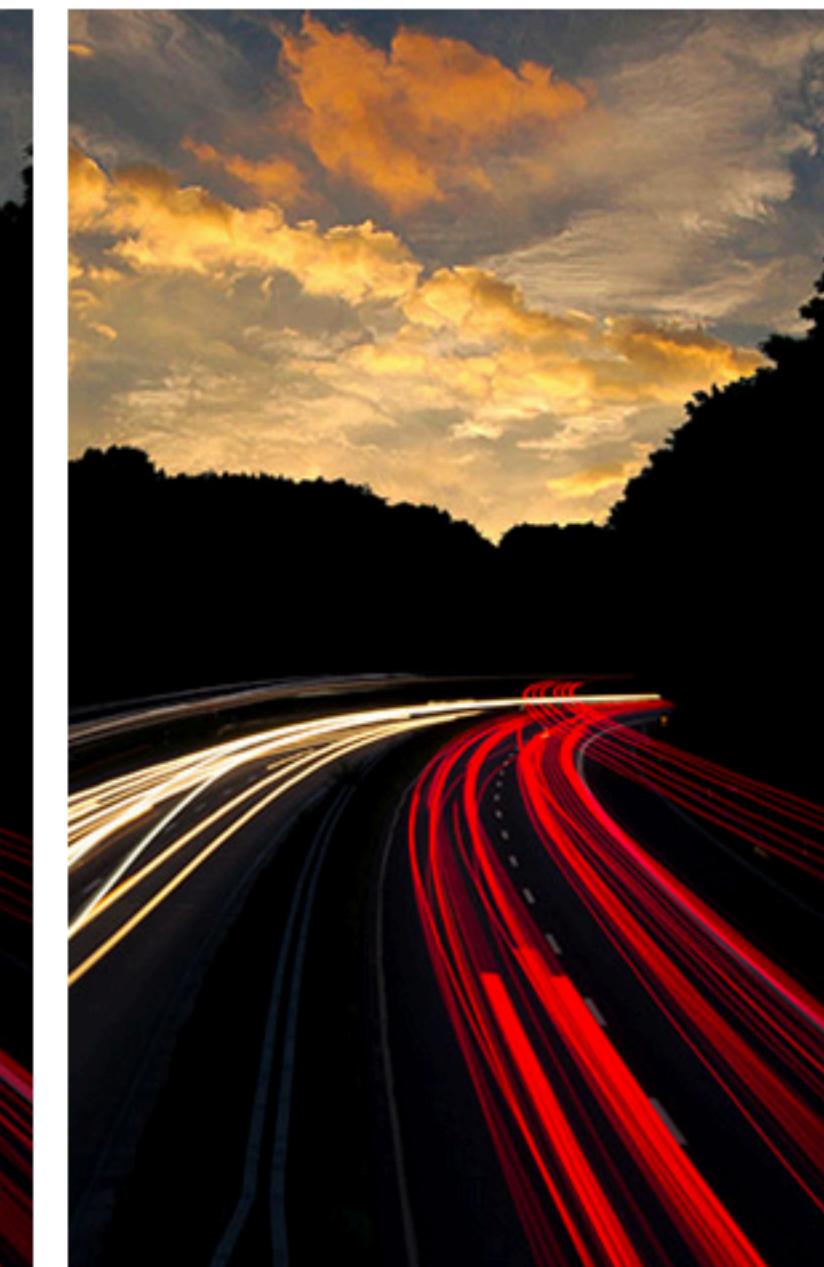
No Positioning



Floated Layout



Flexbox Enhancements



CSS Grid Layout

Progressive Enhancement Example

Enhancing with Grid.

```
section {  
    display: grid;  
    grid-template-columns: repeat(4, 1fr);  
    grid-gap: 20px;  
    max-width: 1300px;  
}
```

Four levels of enhancement



CSS
Grid
Layout



No
Positioning



Floated
Layout



Flexbox
Enhancements

Progressive Enhancement Example

Floating articles

```
article {  
    float: left;  
    width: 24.25%;  
}  
  
article:not(:last-child) {  
    margin-right: 1%;  
}
```

Progressive Enhancement Example

Overwrite only the properties that actually need overwriting.

```
@supports(display: grid) {  
    article {  
        width: auto;  
    }  
  
    article:not(:last-child) {  
        margin-right: 0;  
    }  
}
```

Four levels of enhancement



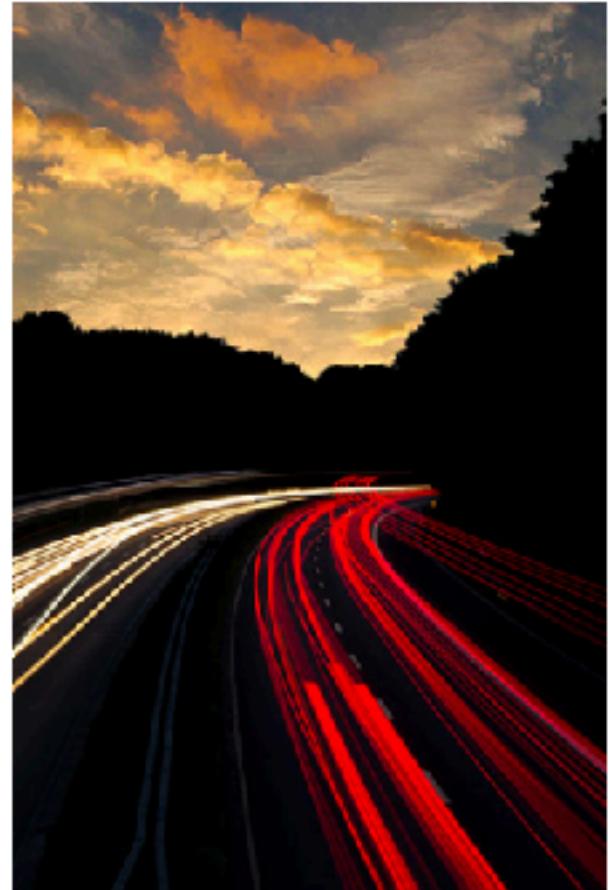
No Positioning



Floated Layout



Flexbox Enhancements



CSS Grid Layout

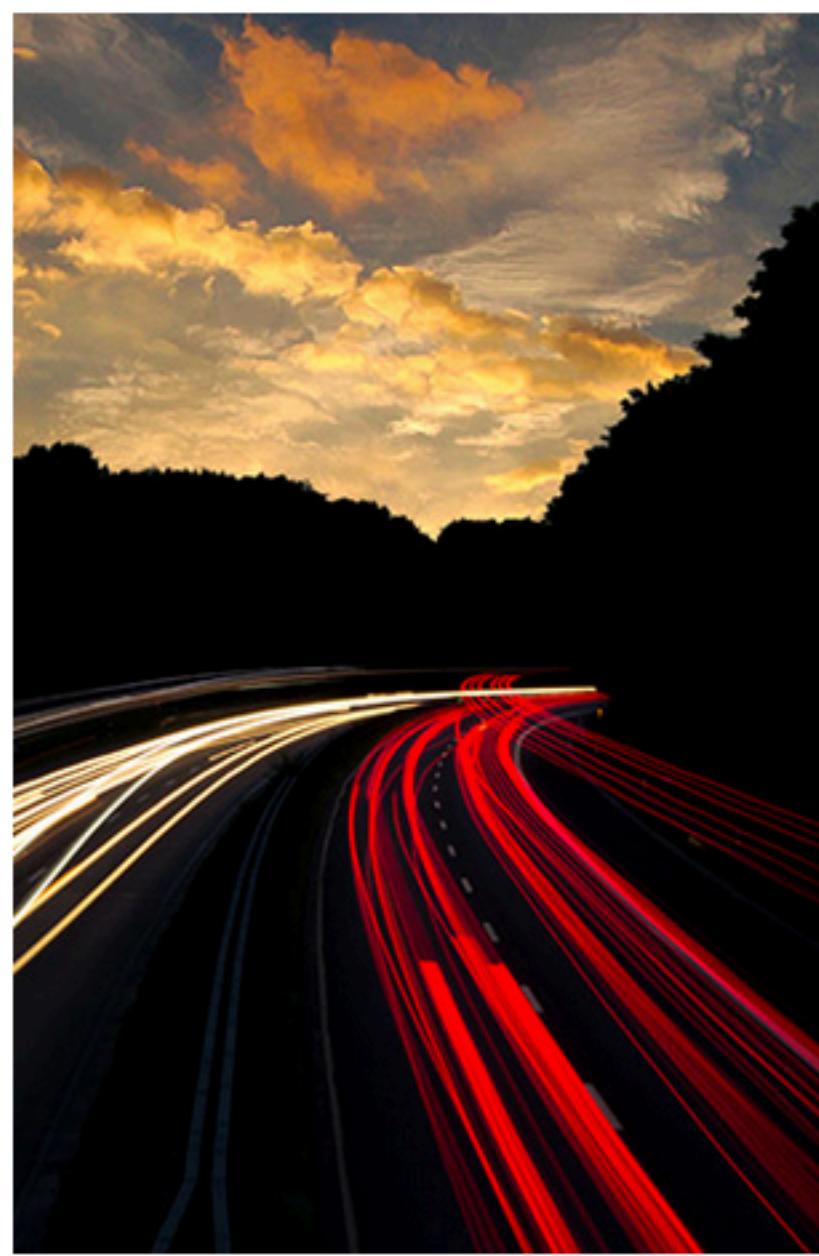
Progressive Enhancement Example

Placing the heading in the second row and aligning the articles.

```
article {  
    float: left;  
    width: 24.25%;  
    display: flex;  
    flex-direction: column;  
    grid-row: span 2;  
}  
  
article:nth-of-type(2) {  
    grid-column: 2;  
    grid-row: 2 / span 2;  
}  
  
h2 {  
    grid-row: 1;  
    grid-column: 2;  
    font-size: 50px;  
    margin: 0;  
}
```

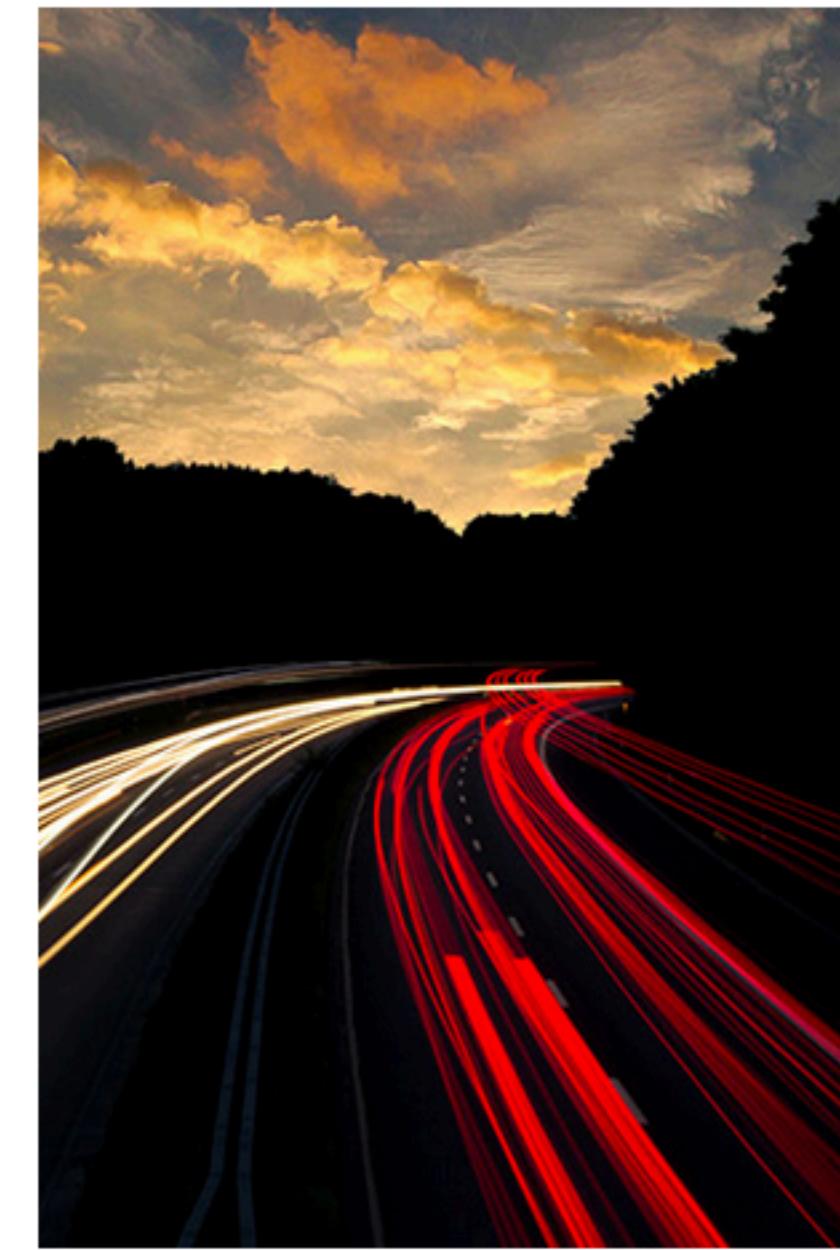


No Positioning

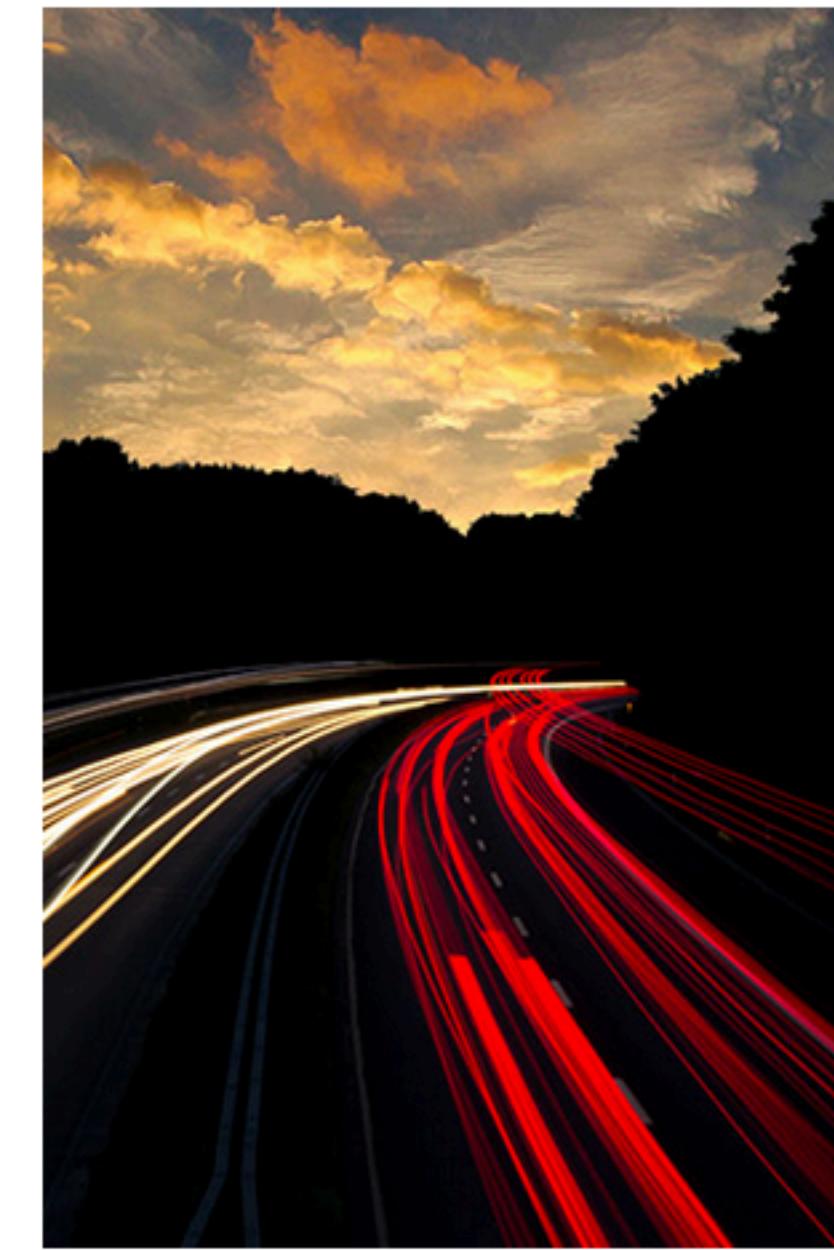


Floated Layout

Four levels of enhancement



Flexbox Enhancements



CSS Grid Layout

CodePen - Demo: Progressive

https://s.codepen.io/metoza/debug/Emdovx

Use Grid today!

[Home](#) [About this demo](#) [Gridbyexample](#) [Progressive Enhancement](#) [Grid Garden](#)

Progressive Enhancement Demo

Leverage the power of CSS Grid Layout!
Read [this article](#) for details on how to use CSS Grid Layout today.

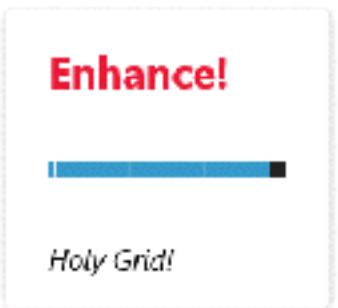
Four levels of enhancement

No Positioning

Floated Layout

Flexbox Enhancements

CSS Grid Layout



Use Grid today!

[Home](#) [About this demo](#) [Gridbyexample](#) [Progressive Enhancement](#) [Grid Garden](#)

Progressive Enhancement Demo

Leverage the power of CSS Grid Layout!

Read [this article](#) for details on how to use CSS Grid Layout today.

Four levels of enhancement



No Positioning

Floated Layout

Flexbox Enhancements

CSS Grid Layout

Browser Support

Works even in very old browsers

Enhance!

Holy Grid!

Use Grid today!

[Home](#) [About this demo](#) [Gridbyexample](#) [Progressive Enhancement](#) [Grid Garden](#)

Progressive Enhancement Demo

Leverage the power of CSS Grid Layout!
Read [this article](#) for details on how to use CSS Grid Layout today.

Four levels of enhancement

No Positioning



Floated Layout



Flexbox Enhancements



CSS Grid Layout



Browser Support

Use Grid today!

[Home](#) [About this demo](#) [Gridbyexample](#) [Progressive Enhancement](#) [Grid Garden](#)

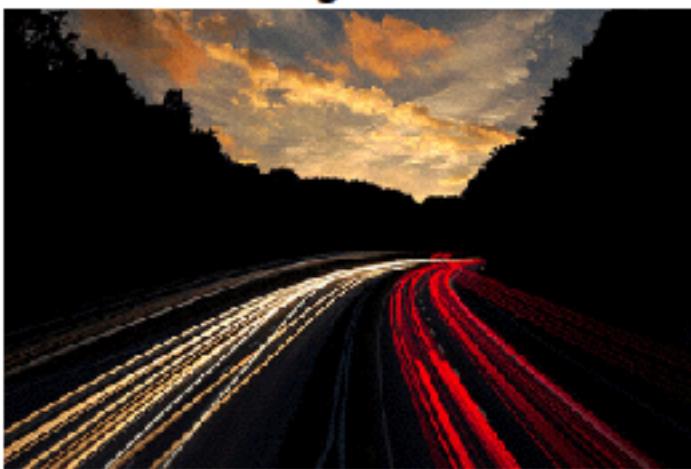
Progressive Enhancement Demo

Leverage the power of CSS Grid Layout today!

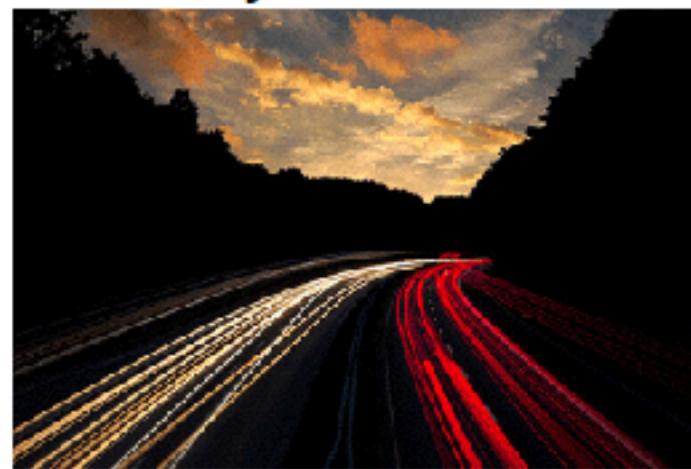
Read [this article](#) for details on how to use CSS Grid Layout today.

Four levels of enhancement

No Positioning



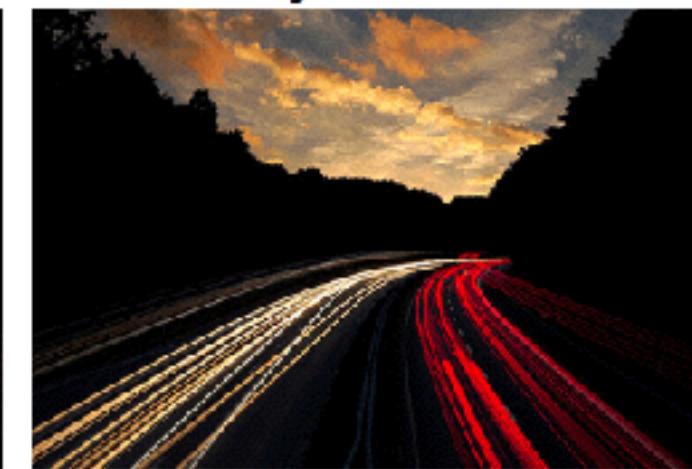
Floated Layout

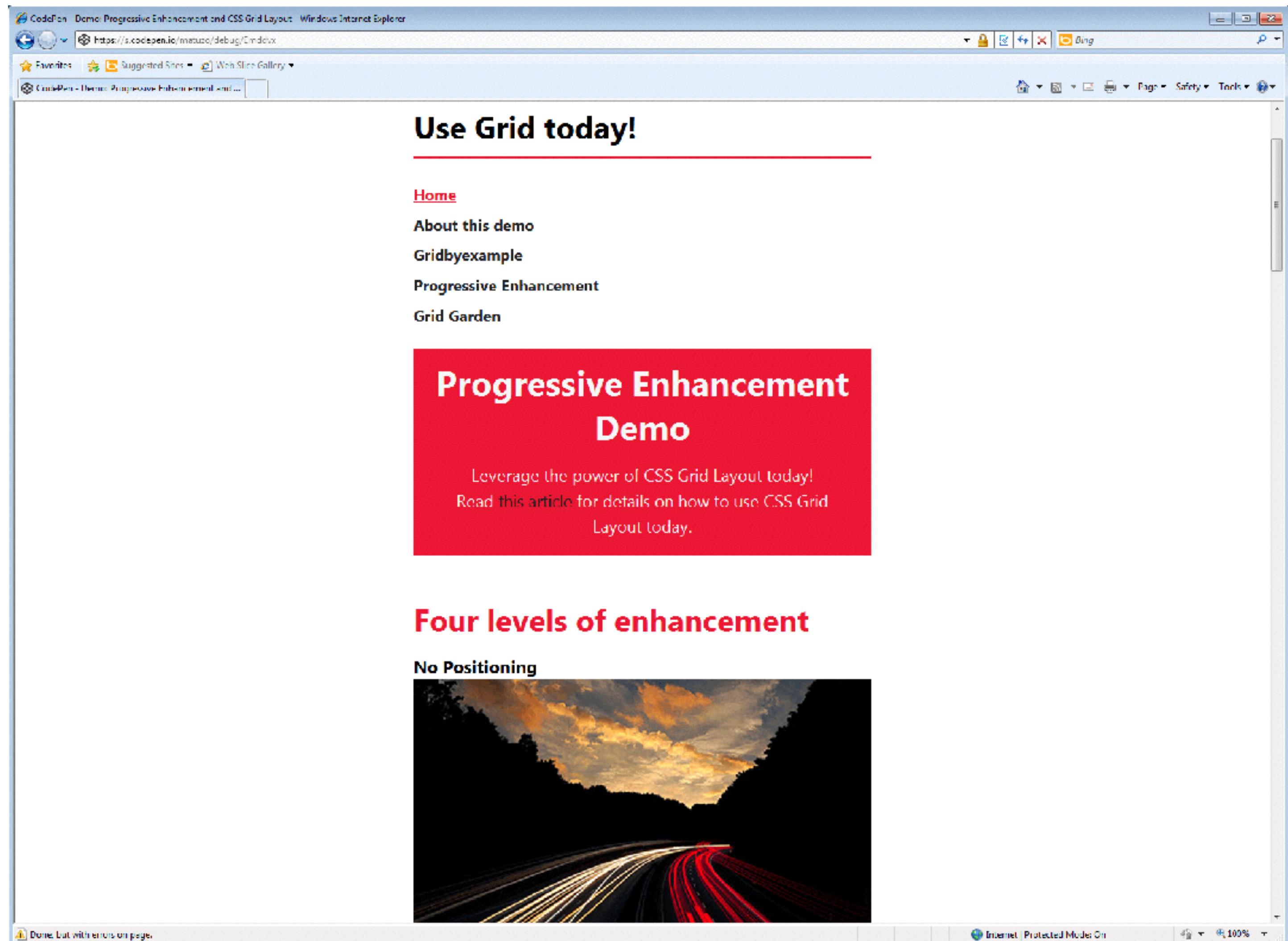


Flexbox Enhancements



CSS Grid Layout





Progressive Enhancement

- Use vendor prefixes only sparingly, if ever.
- Grid is build in a way to coexist with other properties.
- `@support() {}` works in all browsers that support grid.
- It's not necessary to create multiple layouts.
- Please! Don't use poly fills.

Grid properties in IE10 - Edge 15

Level 1 Property	IE 10 Implementation
grid-template-columns	-ms-grid-columns
grid-template-rows	-ms-grid-rows
grid-template-areas	X
grid-template	X
grid-auto-columns	X
grid-auto-rows	X
grid-auto-flow	X

Grid properties in IE10 - Edge 15

Level 1 Property	IE 10 Implementation
grid	X
grid-row-start	-ms-grid-row
grid-column-start	-ms-grid-column
grid-row-end	X (-ms-grid-row-span)
grid-column-end	X (-ms-grid-column-span)
grid-row	Yes (only start value)
grid-column	Yes (only start value)

Grid properties in IE10 - Edge 15

Level 1 Property	IE 10 Implementation
grid-area	X
grid-row-gap	X
grid-column-gap	X
grid-gap	X
align-self	-ms-grid-column-align
justify-self	-ms-grid-row-align

Is prefixing grid properties worth it?

- No auto-placement
- No grid-gap
- No areas
- Box alignment not implemented

„...you can't simply run Autoprefixer and consider the job done.“

Links

- <https://rachelandrew.co.uk/css/cheatsheets/grid-fallbacks>
- <https://rachelandrew.co.uk/archives/2016/11/26/should-i-try-to-use-the-ie-implementation-of-css-grid-layout>
- <https://hacks.mozilla.org/2016/08/using-feature-queries-in-css/>
- <https://www.smashingmagazine.com/2017/07/enhancing-css-layout-floats-flexbox-grid>

MISC

Nesting, Animation, Bugs & more

A screenshot of a CodePen interface showing a grid layout example. The page title is "Automatic Minimum Size of Grid Items".

The layout consists of four grid items:

- Two grid items on the left, each containing the text "Heading".
- One grid item in the center containing a landscape photograph of a road at sunset.
- One grid item at the bottom containing a solid salmon-colored background.

The right side of the screen displays the code for this layout, divided into three sections: HTML, CSS, and JS.

HTML

```
<div class="block">
  
</div>

<div class="grid">
  <div class="grid-item">
    <h2>Heading</h2>
  </div>
  <div class="grid-item">
    <h2>Heading</h2>
  </div>
  <div class="grid-item">
    <h2>Heading</h2>
  </div>
</div>
```

CSS

```
.block {
  background-color: salmon;
  margin-bottom: 20px;
  display: none;
}

.grid {
  display: grid;
  grid-template-columns: 1fr 2fr 1fr;
  grid-gap: 20px;
  width: 90%;
  max-width: 1300px;
  height: 500px;
}

.grid-item {
  background-color: salmon;
  min-width: 0;
}
```

JS

```
last saved less than a minute ago
```

AUTOMATIC MINIMUM SIZE OF GRID ITEMS

bit.ly/grid_sizing

Automatic Minimum Size of Grid Items

- By default, a grid item cannot be smaller than the size of its content.
- Grid items have an initial size of `min-width: auto` and `min-height: auto`.
- That applies to flex items as well.
- <https://codepen.io/matuzo/pen/NXEvor>

Nesting

- A grid item can also be a grid container.
- Parent grid properties don't apply for nested grid items.
- **subgrid** will probably come with level 2 of the specification.
- Workaround for **subgrid**: `display: contents` (<http://gridbyexample.com/video/subgrid-display-contents/>)

Animation

- There are 5 animatable grid properties:
 - `grid-gap`, `grid-row-gap`, `grid-column-gap` as length, percentage, or calc.
 - `grid-template-columns`, `grid-template-rows` as a simple list of length, percentage, or calc, provided the only differences are the values of the length, percentage, or calc components in the list.
 - [Blog post](#) about CSS Grid animation.

Browser	grid-gap, grid-row-gap, grid-column-gap	grid-template-columns	grid-template-rows
Firefox 55, Firefox 53 Mobile	✓	✗	✗
Safari 11.0.2	✗	✗	✗
Chrome 65	✗	✗	✗
Chrome for Android 63, Opera Mini 32	✗	✗	✗
Edge 16	✓	✗	✗

A screenshot of a CodePen editor window titled "CSS Grid Layout: Animation". The page content displays a grid of 12 red rectangular elements labeled "Element 1" through "Element 12" arranged in four rows of three. A blue "ANIMATE" button is positioned above the first row. The right side of the screen shows the code editor with sections for HTML, CSS, and JS.

HTML

```
<p>Check <a href="https://codepen.io/matzko/post/animating-css-grid-layout-properties">this post</a> for more details</p>
<button class="js-button">Animate</button>
<div class="grid js-grid">
  <article class="item">
    <h2>Element 1</h2>
  </article>
  <article class="item">
    <h2>Element 2</h2>
  </article>
  <article class="item">
    <h2>Element 3</h2>
  </article>
  <article class="item">
    <h2>Element 4</h2>
  </article>
  <article class="item">
    <h2>Element 5</h2>
  </article>
  <article class="item">
    <h2>Element 6</h2>
  </article>
  <article class="item">
    <h2>Element 7</h2>
  </article>
  <article class="item">
    <h2>Element 8</h2>
  </article>
  <article class="item">
    <h2>Element 9</h2>
  </article>
  <article class="item">
    <h2>Element 10</h2>
  </article>
  <article class="item">
    <h2>Element 11</h2>
  </article>
  <article class="item">
    <h2>Element 12</h2>
  </article>
</div>
```

CSS

```
.grid {
  display: grid;
  grid-template-columns: 200px 200px 200px;
  grid-template-rows: 100px;
  grid-gap: 20px;
  transition: all 1s;
}

.grid--full {
  grid-template-columns: 300px 300px 300px;
  grid-template-rows: 200px;
  grid-gap: 10px;
}

button {
  background-color: #123456;
  color: #ffffff;
  margin: 20px 0;
}
```

JS

```
document.querySelector('.js-button').addEventListener('click', function()
{
  console.log('test')
  document.querySelector('.js-grid').classList.toggle('grid--full')
})
```

ANIMATION

bit.ly/grid_animation

CSS Grid Layout: Animating grid-gap (FF Only)

A PEN BY [Manuel Matuzovic](#)

Save Fork Settings Change View

HTML

```
<div class="intro">
  <div class="sides">
    <div class="side monkey">
      <h2 class="name">Monkey</h2>
      <div class="emoji">&#128000;</div>
    </div>
    <div class="versus">
      <span>vs.</span>
    </div>
    <div class="side robot">
      <h2 class="name">Robot</h2>
      <div class="emoji">&#128001;</div>
    </div>
  </div>
</div>
```

CSS

```
.sides {
  animation: 0.7s curtain cubic-bezier(.86,0,.07,1) 0.4s both;
}

@keyframes curtain {
  0% {
    grid-gap: 100vw;
  }
  100% {
    grid-gap: 0;
  }
}
```

JS

```
Last saved 6 months ago
```

Console Assets Comments

ANIMATION

bit.ly/grid_animation2

The screenshot shows a GitHub repository page for 'GridBugs' at <https://github.com/rachelandrew/gridbugs>. The page is a README.md file. It starts with a heading 'GridBugs' and a note about being inspired by Flexbugs. It discusses issues related to the Grid Specification and encourages users to post issues if they find them. A section titled 'This is not CSS Grid technical support' provides links to resources like Grid by Example and an AMA. Below this, a section titled 'The bugs' lists 14 specific bugs, each with a detailed description and a table showing browser compatibility. The bugs listed are:

- grid-auto-rows and grid-auto-columns should accept a track listing
- Repeat notation should accept a track listing
- Fragmentation support
- Sizing of items with an intrinsic size inside fixed size grid tracks
- Items with an intrinsic aspect ratio should align start
- The grid-gap property should accept percentage values
- Grid gaps should behave as auto-sized tracks?
- Setting max-height to a percentage should scale down a larger image inside a grid track
- min-content and max-content in track listings
- Some HTML elements can't be grid containers
- A text area that is a grid item collapses to zero width
- Space distributed to empty tracks spanned by an item with content
- Inconsistency with percentage padding and margins
- fit-content is stretching

For the first bug, the table shows:

Demos	Browsers affected	Tracking bugs	Tests
1.1 — bug	Firefox	Firefox #1339672	WPT

The page also includes a note about implicit grid track sizing from the specification.

Is there a grid system for CSS Grid Layout?

- Grid is a grid system!
- You don't even need extra markup for defining columns or rows.
- It might make sense to use mixins in order to enforce a specific style or set of techniques.

The screenshot shows a CodePen experiment titled "Experiment: Pseudo elements as grid items". It compares two layout methods: Grid and Flexbox. Both sections contain four items labeled item 0, item 1, item 2, and item 3. In the Grid section, item 0 is a green box, item 1 is a red box, item 2 is a red box, and item 3 is a green box. In the Flexbox section, item 0 is a green box, item 1 is a red box, item 2 is a red box, and item 3 is a green box. The experiment uses CSS pseudo-elements :before and :after to create additional items within the grid structure.

Grid

item 0 item 3 item1 item2

Flexbox

item 0 item1 item2 item 3

HTML

```
<h2>Grid</h2>
<section class="grid">
  <div class="item">item1</div>
  <div class="item">item2</div>
</section>

<h2>Flexbox</h2>
<section class="flex">
  <div class="item">item1</div>
  <div class="item">item2</div>
</section>
```

CSS

```
.grid {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr 1fr;
  grid-template-rows: 100px 100px;
  grid-gap: 20px;
}

.flex {
  display: flex;
  justify-content: space-between;
}

section:before {
  content: "item 0";
  display: block;
  background-color: #7CCB80;
  border-radius: 5px;
  padding: 10px;
}

section:after {
  content: "item 3";
  display: block;
  grid-column: 2;
  grid-row: 1 / span 2;
  background-color: #7CCB80;
  border-radius: 5px;
  padding: 10px;
}
```

JS

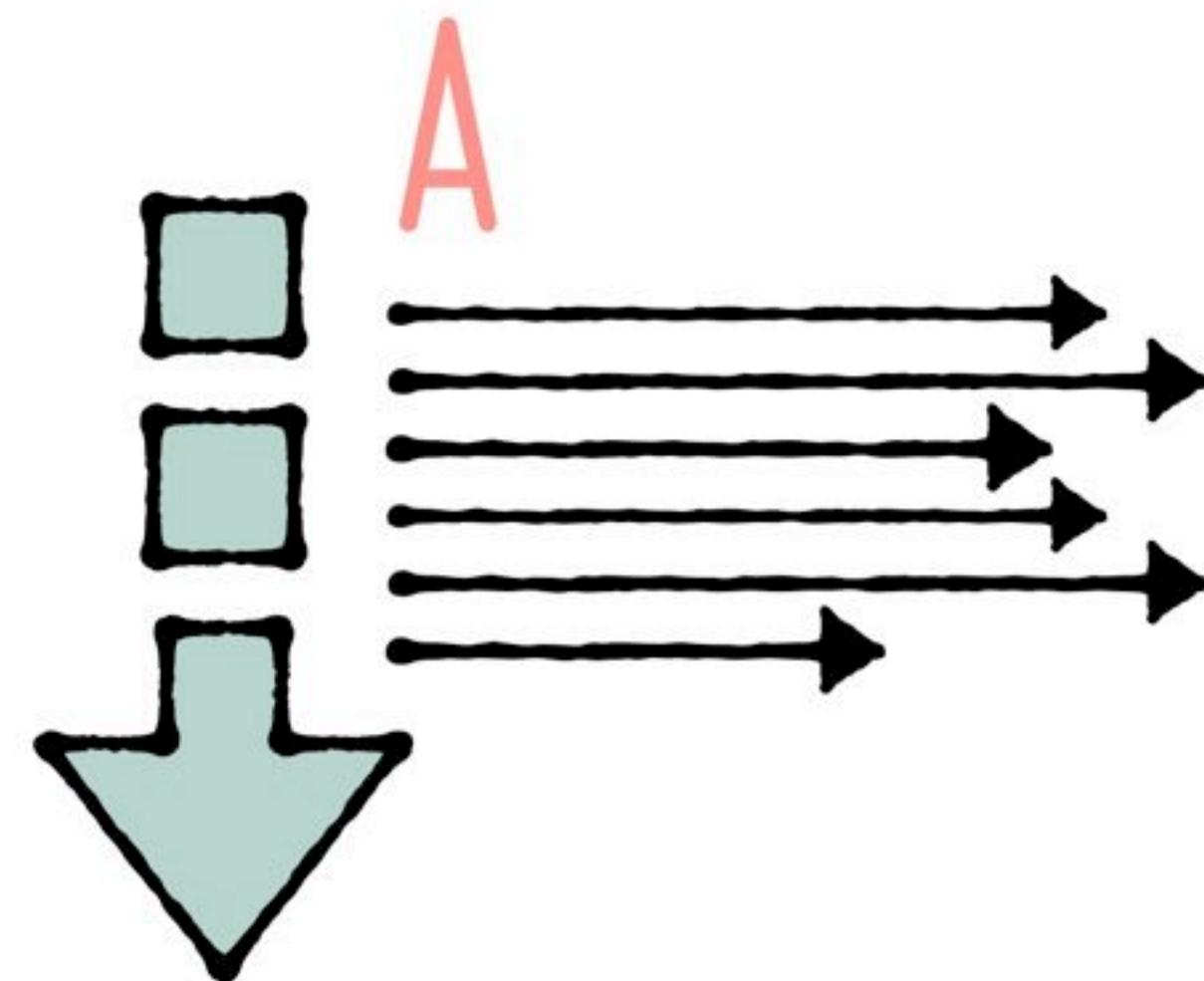
```
last saved 7 months ago
```

PSEUDO ELEMENTS

bit.ly/grid_pseudo



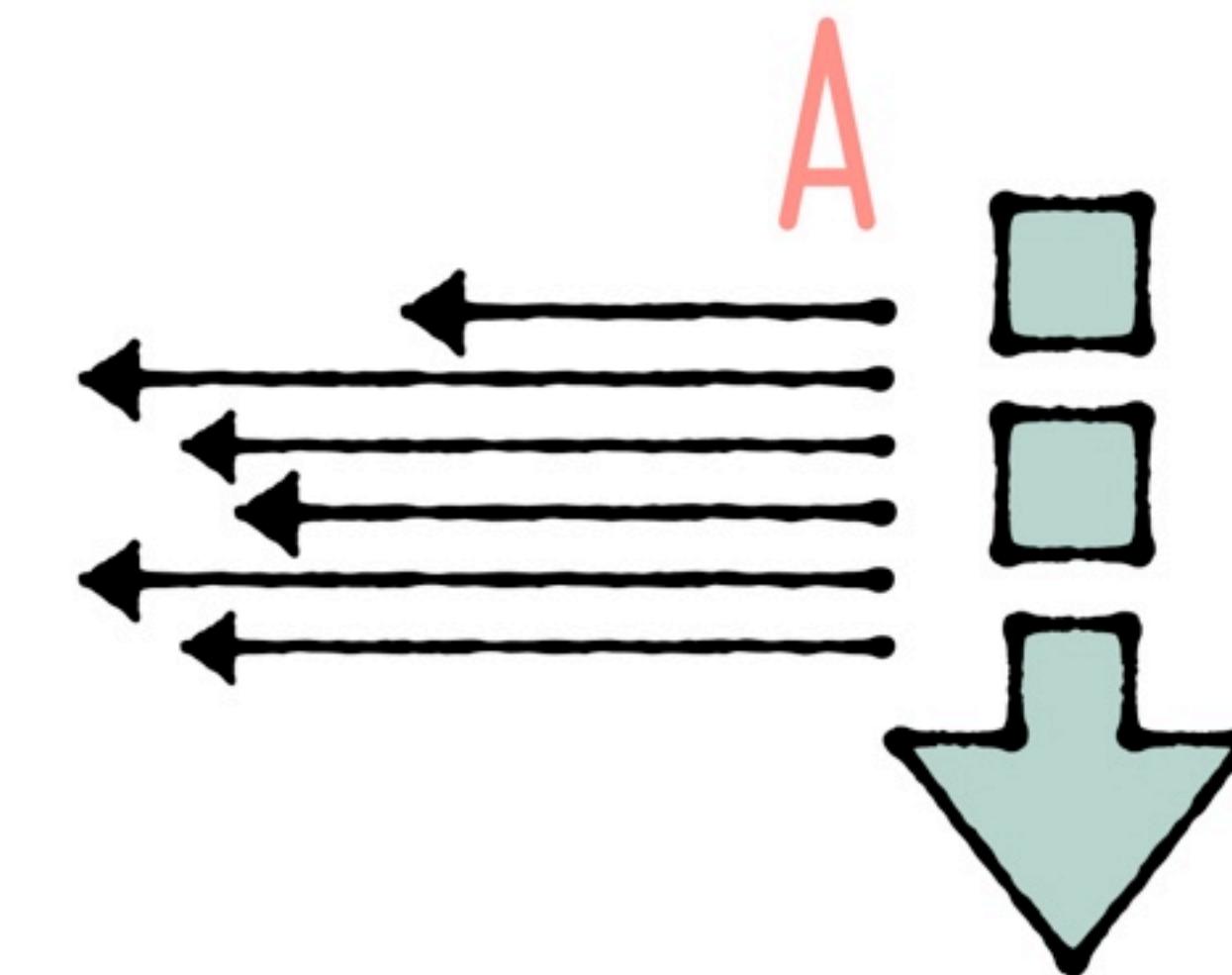
LATIN-BASED SYSTEMS





WRITING MODE: ARABIC-BASED SYSTEMS

ARABIC-BASED SYSTEMS



The screenshot shows a CodePen editor window titled "CSS Grid Layout: rtl direction". The page content displays five red rectangular elements labeled "Element 1" through "Element 5" arranged in two rows. The first row contains "Element 1" (left), "Element 2" (center), and "Element 3" (right). The second row contains "Element 4" (left) and "Element 5" (right). The browser's developer tools are open, showing the HTML and CSS code.

```
HTML
1 <div class="grid">
2   <article class="item">
3     <h2>Element 1</h2>
4   </article>
5   <article class="item">
6     <h2>Element 2</h2>
7   </article>
8   <article class="item">
9     <h2>Element 3</h2>
10  </article>
11  <article class="item">
12    ...
13  </article>
14  <article class="item">
15    ...
16  </article>
17  <article class="item">
18    ...
19  </article>
20  <article class="item">
21    ...
22  </article>
23  <article class="item">
24    ...
25  </article>
26  <article class="item">
27    ...
28  </article>
29  <article class="item">
30    ...
31  </article>
32  <article class="item">
33    ...
34  </article>
35  <article class="item">
36    ...
37  </article>
38  <article class="item">
39    ...
40  </article>
41  <article class="item">
42    ...
43  </article>
44  <article class="item">
45    ...
46  </article>
47  <article class="item">
48    ...
49  </article>
50  <article class="item">
51    ...
52  </article>
53  <article class="item">
54    ...
55  </article>
56  <article class="item">
57    ...
58  </article>
59  <article class="item">
60    ...
61  </article>
62  <article class="item">
63    ...
64  </article>
65  <article class="item">
66    ...
67  </article>
68  <article class="item">
69    ...
70  </article>
71  <article class="item">
72    ...
73  </article>
74  <article class="item">
75    ...
76  </article>
77  <article class="item">
78    ...
79  </article>
80  <article class="item">
81    ...
82  </article>
83  <article class="item">
84    ...
85  </article>
86  <article class="item">
87    ...
88  </article>
89  <article class="item">
90    ...
91  </article>
92  <article class="item">
93    ...
94  </article>
95  <article class="item">
96    ...
97  </article>
98  <article class="item">
99    ...
100 </article>
101 </div>
```

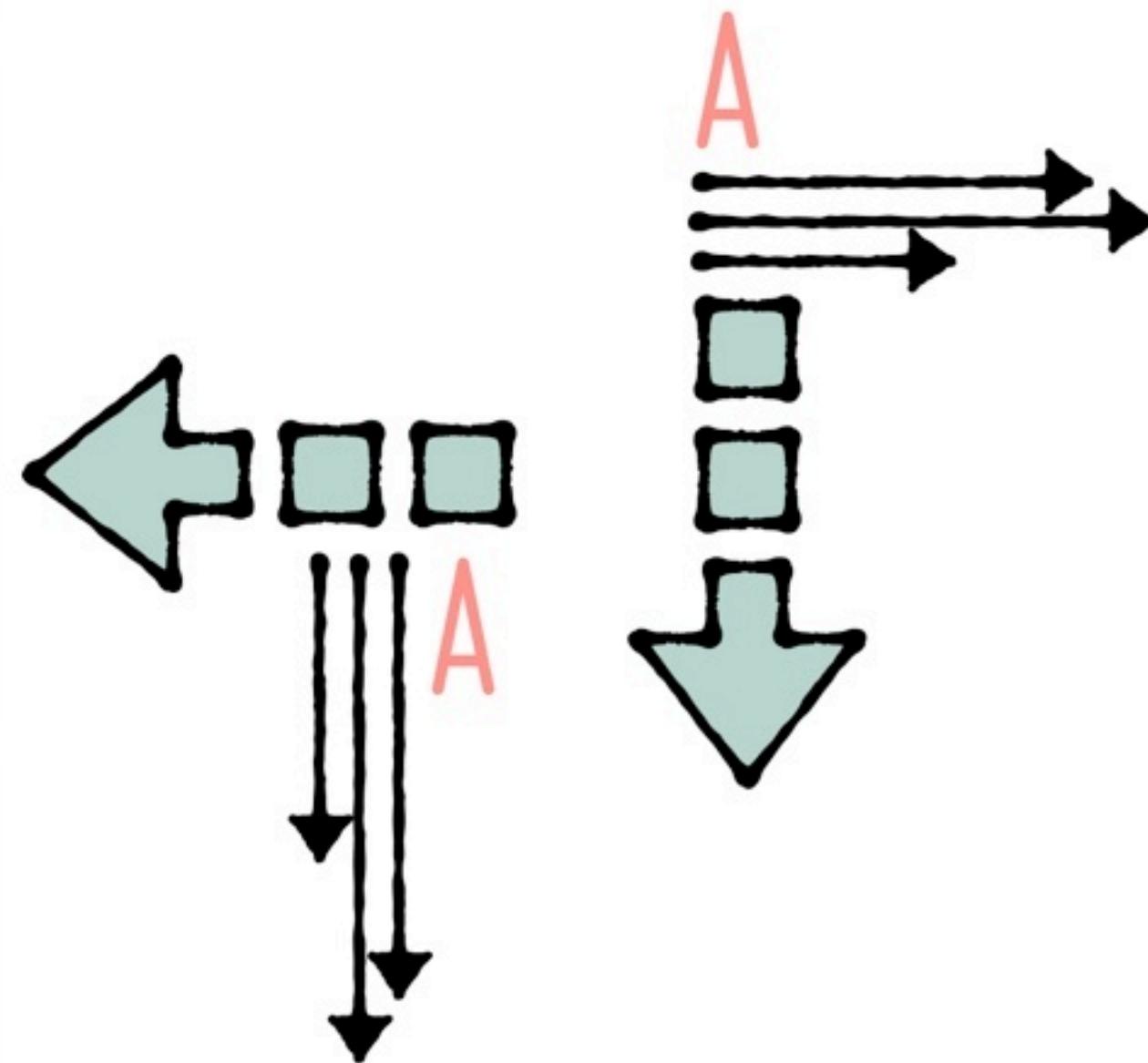
```
CSS
1 html {
2   direction: rtl;
3 }
4
5 .grid {
6   height: 500px;
7
8   display: grid;
9   grid-template-columns: 20% 20% 20%;
10  grid-gap: 20px;
11  justify-content: end;
12 }
13
14 .item:First-child {
15   justify-self: end;
16 }
17
18 .item:nth-child(2) {
19   justify-self: start;
20 }
```

```
JS
```



HAN-BASED SYSTEMS

*Chinese, Japanese, Korean & more



The screenshot shows a CodePen editor window with the title "CSS Grid Layout: vertical-rl writing-mode". The page content displays five red rectangular elements labeled "Element 1" through "Element 5" arranged in a grid. A large red border surrounds the entire grid area. The CodePen interface includes tabs for "HTML", "CSS (SCSS)", and "JS", along with various toolbars and status messages.

```
HTML
1 <div class="grid">
2   <article class="item">
3     <h2>Element 1</h2>
4   </article>
5   <article class="item">
6     <h2>Element 2</h2>
7   </article>
8   <article class="item">
9     <h2>Element 3</h2>
10  </article>
11  <article class="item">
12    <h2>Element 4</h2>
13  </article>
14  <article class="item">
15    <h2>Element 5</h2>
16  </article>
17</div>
```

```
CSS (SCSS)
html {
  writing-mode: vertical-rl;
}

.grid {
  border: 10px solid red;
  display: grid;
  grid-template-columns: 200px 200px 200px;
  grid-template-rows: 80px 80px 80px;
  grid-gap: 20px;
  justify-content: end;
}

.item:first-child {
  justify-self: end;
}

.item:nth-child(2) {
  justify-self: start;
}
```

A screenshot of a CodePen editor window titled "CSS Grid Layout: Anonymous items". The URL in the address bar is <https://codepen.io/matuzo/pen/GQKpPz?editors=1100>. The page content displays the text "CSS Grid Layout is awesome" in a grid layout. The CodePen interface includes sections for HTML, CSS, and JS, along with navigation and sharing tools.

HTML

```
<div class="grid">
<strong>CSS Grid Layout</strong> is
<strong>awesome</strong>.
</div>
```

CSS

```
.grid {
  display: grid;
  grid-template-columns: repeat(4, 1fr);
  font-size: 2rem;
  font-family: sans-serif;
}
```

JS

Last saved less than a minute ago [Edit](#) [Delete](#) [Share](#) [Export](#) [Embed](#) [Collections](#)

ANONYMOUS ITEMS

bit.ly/grid_anon

WHAT'S NEXT?

Nesting, Debugging, RWD

Gaps

- `grid-column-gap`, `grid-row-gap` and `grid-gap` will be renamed to `column-gap`, `row-gap` and `a gap`.
- It is likely that browsers will alias the old names to the new for the foreseeable future.
- The new gap properties are going to be part of the Box Alignment specification.
- They're not limited to Grid. Available to other layout modes where they make sense (Flexbox!).

Subgrid

- Only direct children of a grid container become grid items. Children of grid items are no affected.
- It's possible to nest grids, but the child grids have no relationship to the parent.
- `subgrid` has been deferred to Level 2 due to lack of implementation and desire for further discussion.

display:contents

- The element itself does not generate any boxes, but its children and pseudo-elements still generate boxes as normal.
- Behaves as if it had been replaced with its children. Child elements take up its place in the document tree.
- Could be used in order to simulate `display: subgrid`.

This screenshot shows a browser-based code editor interface, likely from CodePen, displaying a live preview and the source code for a demonstration of the `display: contents` CSS property.

Live Preview:

The preview window shows a red-bordered box containing two paragraphs of text. The first paragraph explains what happens when `display: contents` works, resulting in a full-width box with a red border. The second paragraph explains what happens if it doesn't work or if the `display` property is removed from the `.content` selector, resulting in a 400px wide box with a grey border and background color, nested inside the red-bordered box.

HTML:

```
<div class="content item">
  <div class="inner">
    <p>This is the inner box. If display: contents works in your browser you will see a full width box with a red border.</p>
    <p>If display: contents does not work or if you remove the display property from .content you will see a 400 pixel box with a grey border and background color, inside will be nested the box with the red border.</p>
  </div>
</div>
```

CSS:

```
.content {
  border: 2px solid #D73038;
  border-radius: 5px;
  background-color: #D73038;
  padding: 10px;
  width: 400px;
}

.inner {
  border: 2px solid #17242D;
  border-radius: 5px;
  padding: 10px;
}

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

JS:

```
console.log('Hello, world!');
```

Toolbar:

Save, Fork, Settings, Change View, etc.

Last saved less than a minute ago, Delete, Share, Export, Embed, Collections.

DISPLAY: CONTENTS

bit.ly/grid_contents2

display: contents (FF only)

Fake subgrid

```
HTML
6+ <li class="item"><a href="#">Element 2</a></li>
7+ <li class="item"><a href="#">Element 3</a></li>
8+ <li class="item"><a href="#">Element 4</a></li>
9+ <li class="item"><a href="#">Element 5</a></li>
10+ <li class="item"><a href="#">Element 6</a></li>
11+ </ul>
12+ </div>
13+
14+ <h2>Fake subgrid</h2>
15+ <div class="wrapper subgrid">
16+   <h3>Item</h3>Heading</h2>
17+   <ul>
18+     <li class="item"><a href="#">Element 1</a></li>
19+     <li class="item"><a href="#">Element 2</a></li>
20+     <li class="item"><a href="#">Element 3</a></li>
21+     <li class="item"><a href="#">Element 4</a></li>
22+     <li class="item"><a href="#">Element 5</a></li>
23+     <li class="item"><a href="#">Element 6</a></li>
24+   </ul>
25+ </div>

CSS
1+ .wrapper {
2+   display: grid;
3+   grid-template-columns: 120px repeat(2, 1fr);
4+   grid-gap: 20px;
5+ }
6+
7+ h2 {
8+   grid-column: 2 / -1;
9+ }
10+
11+ .contents ul {
12+   display: contents;
13+ }
14+
15+ .subgrid ul {
16+   /* span the whole grid */
17+   grid-column: 1 / -1;
18+
19+   /* create another grid and inherit the values from the parent grid */
20+   display: inherit;
21+   grid-template-columns: inherit;
22+   grid-gap: inherit;
23+
24+   /* overwrite the display for browsers that understand display: contents */
25+   display: contents;
26+ }
27+
28+
29+
30+
31+
32+
33+
34+
35+
```

JS

DISPLAY: CONTENTS

bit.ly/grid_contents1

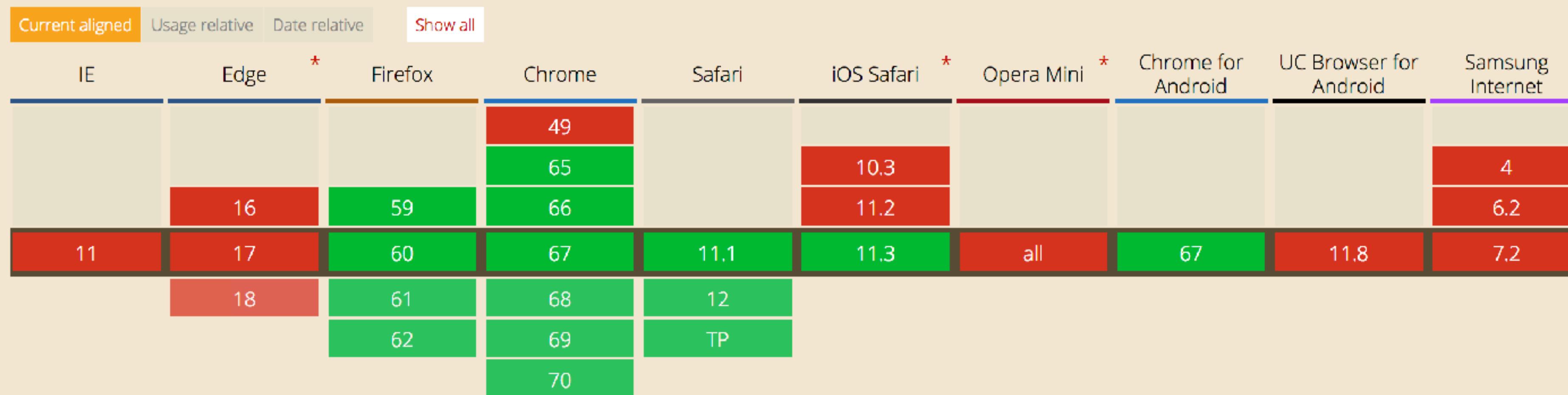
Subgrid with display:contents?

- Issue 1: You cannot apply backgrounds and borders to an item with display: contents.
- Issue 2: Items behave as if they were direct child items, they don't become direct child items. That's why direct child item selectors wouldn't work.
- Issue 3: You lose auto-placement of the parent grid item, which could be undesirable.

Browsersupport

CSS display: contents - WD

`display: contents` causes an element's children to appear as if they were direct children of the element's parent, ignoring the element itself. This can be useful when a wrapper element should be ignored when using CSS grid or similar layout techniques.



`display: contents` is enabled by default in Blink and WebKit and it will be probably shipped in Chrome 65 and Safari 11.1.

Masonry Layout ???

- Grid was designed for the creation of two dimensional grids, which a Masonry Layout isn't.
- Masonry Layouts with Grid are currently only possible with nasty hacks and fixed heights.
- There's an issue in the unofficial level 2 draft which suggests to at least consider implementing a Masonry feature.

Styling cells, tracks and areas

- Styling lines, cells, tracks and areas is currently not possible.
- It would be nice, though.
- Issue on [Github](#).

More

- Creating non-rectangular grid areas.
- Flowing content through grid cells or areas.
- Solving the keyboard/layout disconnect.
- Read [What next for CSS Grid Layout?](#) by Rachel for more details.

WRAP-UP

Wrap-UP

- Use Grid now (when it makes sense). It has been developed for 5 years and it's implemented by all browser vendors. It's not the same disaster as Flexbox.
- Use Grid for the over all page layout as well as on a component level.
- Use it for two-dimensional layouting.
- Leverage the power of progressive enhancement.
- Make sure your grids are accessible.

Wrap-UP

- Explicit grids. (`grid-template-rows` and `grid-template-columns`)
- Implicit grids (`grid-auto-rows` and `grid-auto-columns`)
- Flow (`column`, `row`, `dense`)
- Lines (`grid-column`, `grid-row`, `span keyword`)
- Cells
- Tracks (`minmax`, `repeat`, `auto-fill`, `auto-fit`)
- Areas (`layouts`, `magic lines`, `magic areas`)

Wrap-UP

- Box Alignment Module (justify-items, align-items, justify-self, align-self, justify-content, align-content, grid-gap (gap)).
- MQ breakpoints vs. auto-fill and auto-fit.
- Animation.
- Level 2 Spec.

Links

- <http://gridbyexample.com/>
- <https://css-tricks.com/snippets/css/complete-guide-grid/>
- <https://developer.mozilla.org/en/docs/Web/CSS/grid>
- <https://www.w3.org/TR/css-grid-1/>
- <https://css-tricks.com/getting-started-css-grid/>
- <https://css-tricks.com/collection-interesting-facts-css-grid-layout/>
- <https://css-tricks.com/difference-explicit-implicit-grids/>