

CSS GRID LAYOUT

Manuel
Matuzović
[@mmatuzo](https://twitter.com/mmatuzo) 

CSS GRID LAYOUT

FINALLY!

Overview

Overview

- What is Grid and why do we need it.

Overview

- What is Grid and why do we need it.
- Browser support: Can we use it today?

Overview

- What is Grid and why do we need it.
- Browser support: Can we use it today?
- New properties, units and keywords.

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.

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.

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

Overview

- Common patterns.

Overview

- Common patterns.
- Stacking order.

Overview

- Common patterns.
- Stacking order.
- Responsive Webdesign.

Overview

- Common patterns.
- Stacking order.
- Responsive Webdesign.
- Progressive Enhancement.

Overview

- Common patterns.
- Stacking order.
- Responsive Webdesign.
- Progressive Enhancement.
- Debugging.

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?

What's CSS Grid Layout?

- The first true layout method in CSS.

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.**

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.

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.

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

History and background

- Early 90s: First ideas, but “too complex” to implement.

History and background

- Early 90s: First ideas, but “too complex” to implement.
- 1996 “frame-based” layout model.

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.

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

History and background

- Microsoft shipped a grid layout implementation behind the `-ms-` vendor prefix in Internet Explorer 10 in 2011.

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.

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.

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

History and background

- The media company Bloomberg hired Igalia, an open source consultancy, to implement CSS Grid for both Blink and WebKit.

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.

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.

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

History and background

- Google landed an implementation of CSS Grid in Chromium 56 for Android in January of 2017.

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.

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.

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).

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!

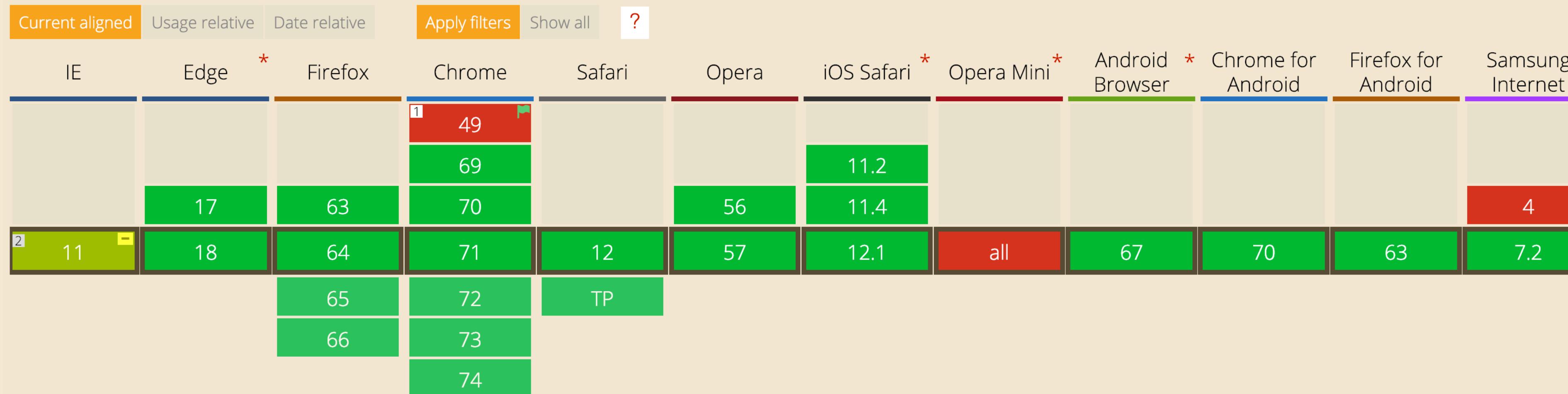
BROWSERSUPPORT?

Browsersupport

CSS Grid Layout - CR

Method of using a grid concept to lay out content, providing a mechanism for authors to divide available space for layout into columns and rows using a set of predictable sizing behaviors.
Includes support for all `grid-*` properties and the `fr` unit.

Usage
Global
unprefixed:
 $85.49\% + 2.85\% = 88.35\%$
 85.49%



Browsersupport

Browsers support

- All major desktop and mobile browsers.

Browsers support

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

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?

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?

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?

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?

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

Is Grid a replacement for Flexbox?

Flexbox

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

Is Grid a replacement for Flexbox?

Flexbox

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

...when child items dictate the layout.

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

Is Grid a replacement for Flexbox?

Grid

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

Is Grid a replacement for Flexbox?

Grid

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

...when the parent item dictates the layout.

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.

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

Grid Basics

LEGEND

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

The grid container.

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

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

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

The screenshot shows a CodePen editor window with the title "CSS Grid Layout Basics: columns, rows, gaps". The page content displays a red grid container containing nine white rectangular items, each labeled "Element 1" through "Element 9". The CodePen interface includes an HTML panel with the following code:

```
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">
```

The CSS panel contains the following rule:

```
.grid {
```

COLUMNS, ROWS & GAPS

bit.ly/grid_basic1

Grid Basics

Defining grids.

Values:

grid

inline-grid

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

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;  
}
```

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;  
}
```

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 CodePen interface with a dark theme. The main content area displays a 3x3 grid of red boxes, each containing a white text label. The grid is defined by the following CSS:

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

The CodePen interface includes tabs for HTML, CSS, and JS, and a sidebar with various tools and settings.

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 area displays a grid of 11 red rectangular boxes containing text. The grid has 3 columns and 4 rows. The first row contains "item 01", "item 04", and "item 10". The second row contains "item 02", "item 05", and "item 11". The third row contains "item 03", "item 06", and "item 09". The fourth row is empty. On the right side of the editor, 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>

```

The "CSS" tab shows the following CSS 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

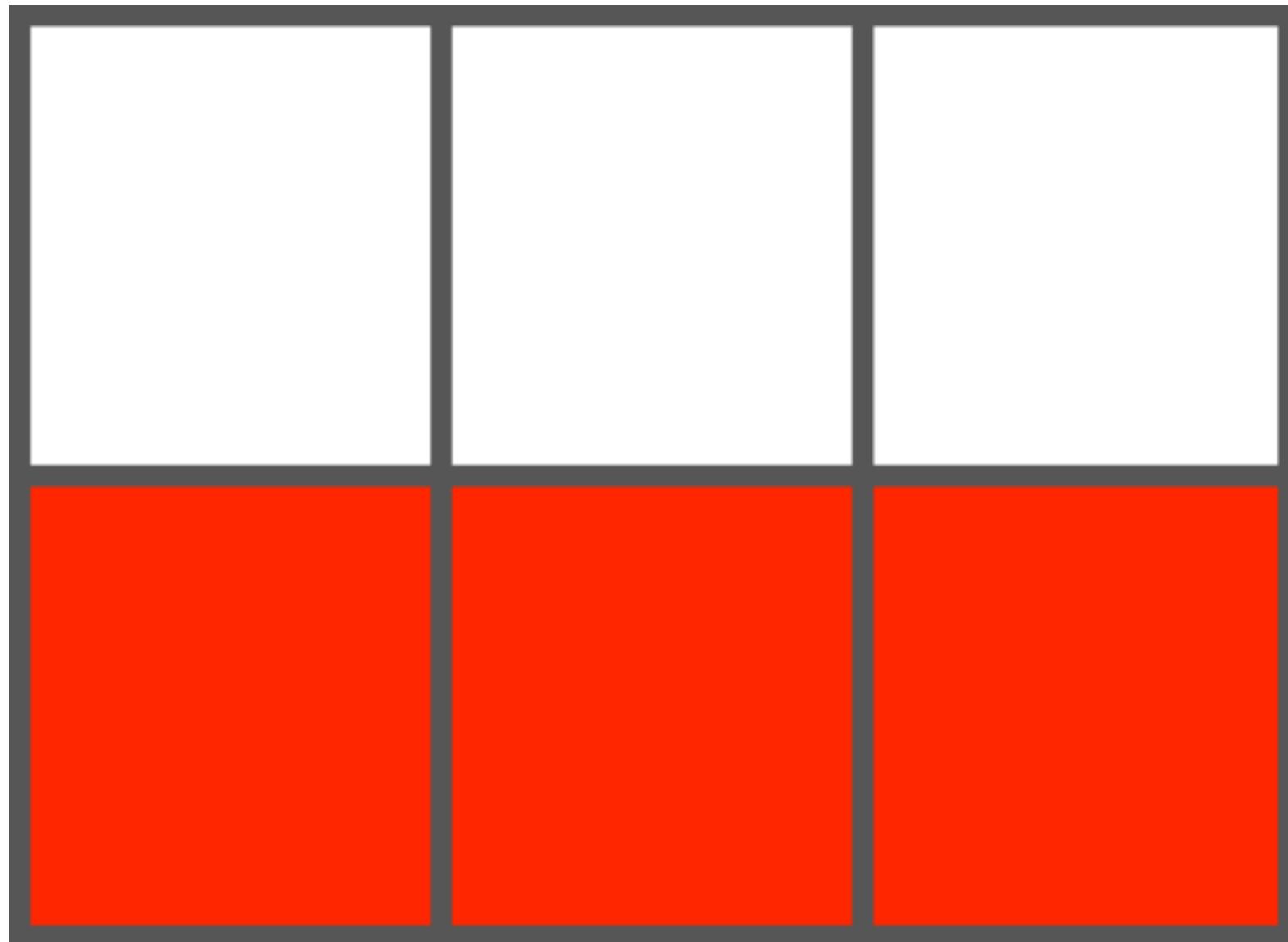
A “dense” packing algorithm, which attempts to fill in holes earlier in the grid if smaller items come up later.

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

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

Terminology

TRACK



Either horizontal or vertical

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

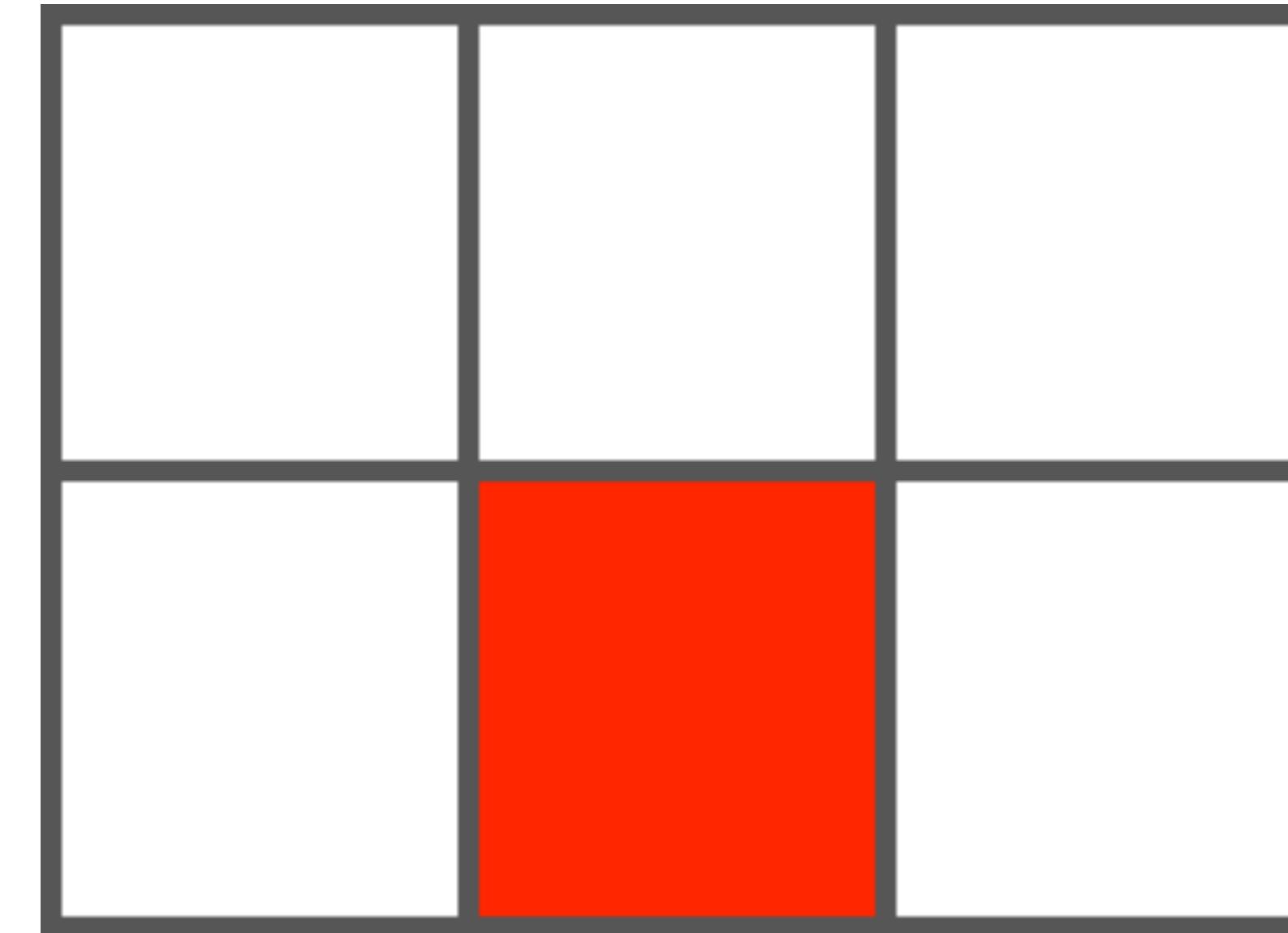
Terminology

TRACK



Either horizontal or vertical

CELL



A single cell within a track

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

The screenshot shows a browser window with three tabs: "Implicit grid", "CSS Grid Layout Basics: column", and "CodePen - CSS Grid Layout". The main content area displays a 3x4 grid of numbered boxes (1-12) and descriptive text blocks. A green grid icon is in the top right corner.

Grid Line	1	2	3	4
Top	1	2	3	4
Left	5	6	7	8
Right	9	10	11	12
Bottom	As you can see, grid tracks can overlap			

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

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

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 CodePen editor window titled "Implicit grid". The URL in the address bar is <https://codepen.io/matuzo/pen/eyXdOa?editors=1100>. The page content displays a horizontal red bar divided into five equal-width segments, each containing a white number from 1 to 5. The CodePen interface includes a sidebar with "HTML", "CSS (SCSS)", and "JS" sections, and a footer with various navigation and status buttons.

```
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>
```

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

IMPLICIT GRIDS

bit.ly/grid_implicit

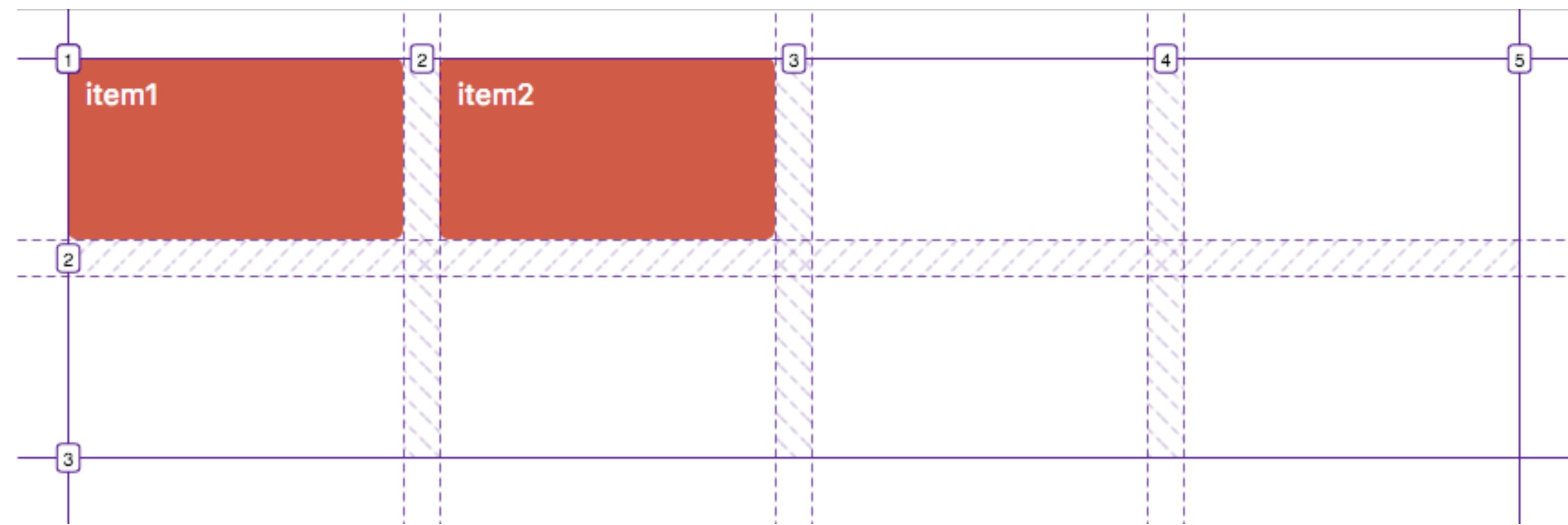
Explicit grids

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`.

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

Implicit grids

- The grid container automatically generates grid tracks by adding grid lines to the grid

Implicit grids

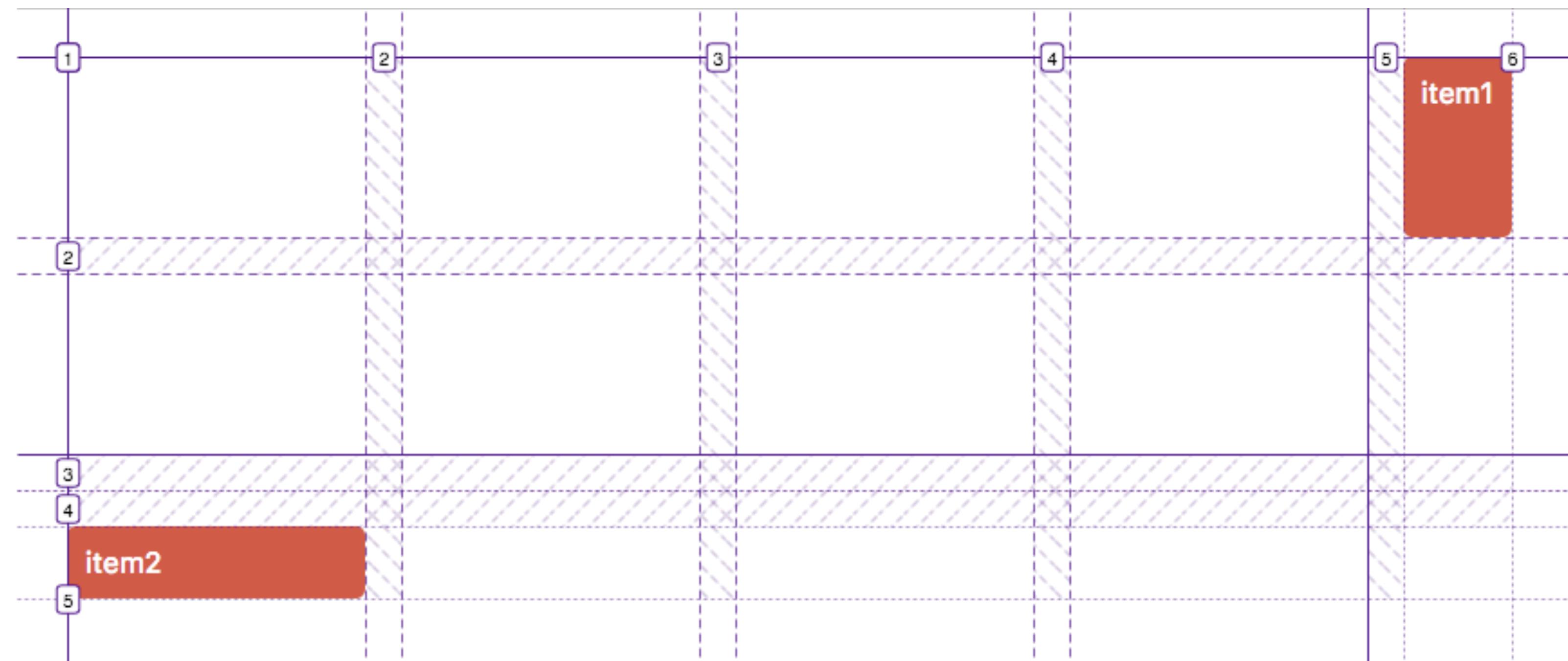
- The grid container automatically generates grid tracks by adding grid lines to the grid
 - ... if there are more grid items than cells in the grid

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.

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". The main preview area displays a grid of five red rectangular boxes labeled 1 through 5. Box 1 is in the top-left position, Box 2 is to its right, Box 3 is below Box 1, Box 4 is to the right of Box 3, and Box 5 is positioned below Box 4. The grid has a total of 2 columns and 3 rows. The CSS panel contains the following SCSS code:

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

The HTML panel shows the generated HTML code:

```
<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>
```

The bottom navigation bar includes links for Console, Assets, Comments, and a file icon. It also shows a message "Last saved less than a minute ago" and buttons for Delete, Share, Export, Embed, and Collections.

SIZING IMPLICIT GRIDS

bit.ly/grid_implicit2

Grid Basics

Sizing implicit columns

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

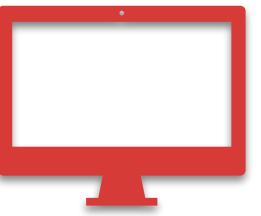
Grid Basics

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

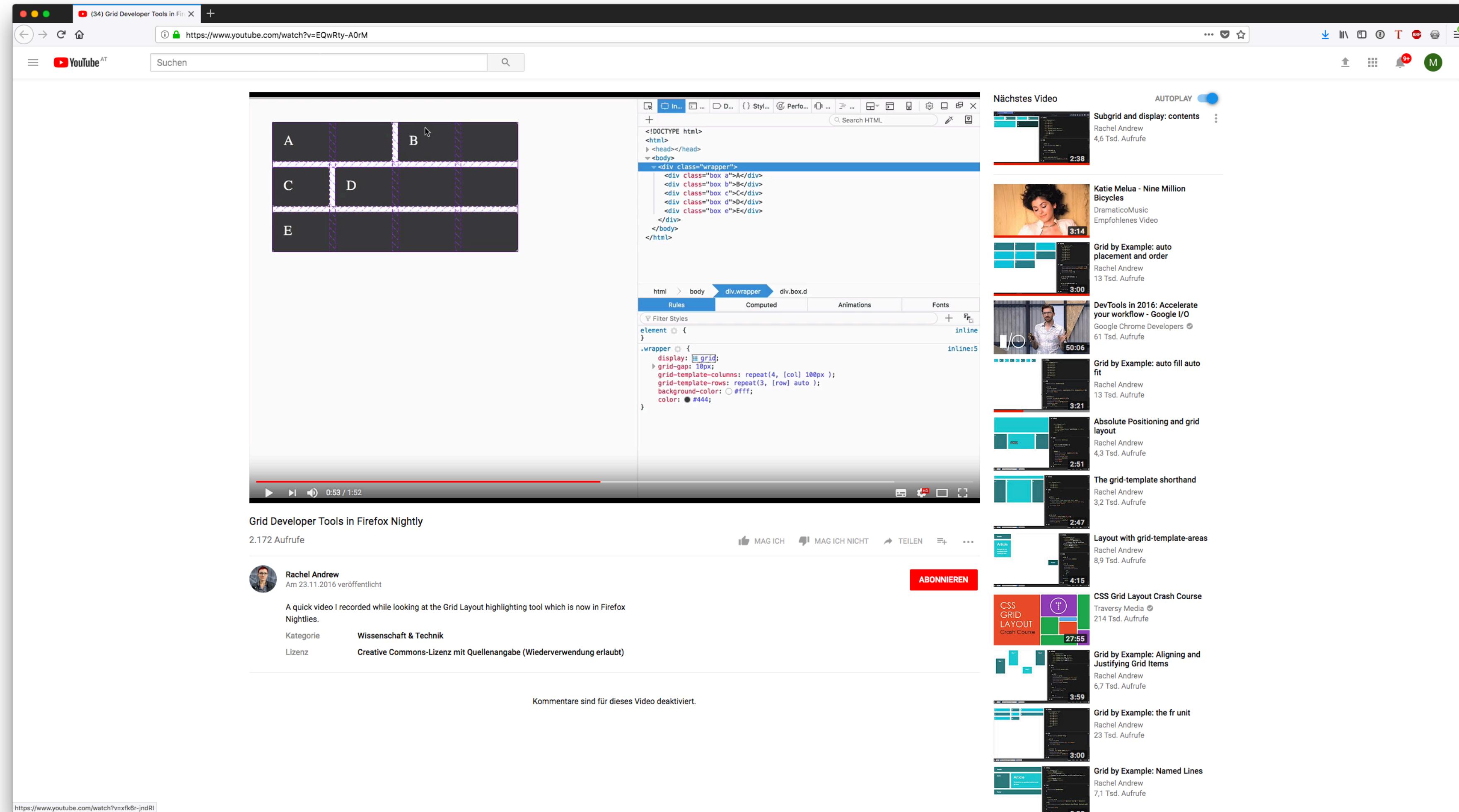
Debugging

Firefox is currently the best browser for debugging.

Debugging

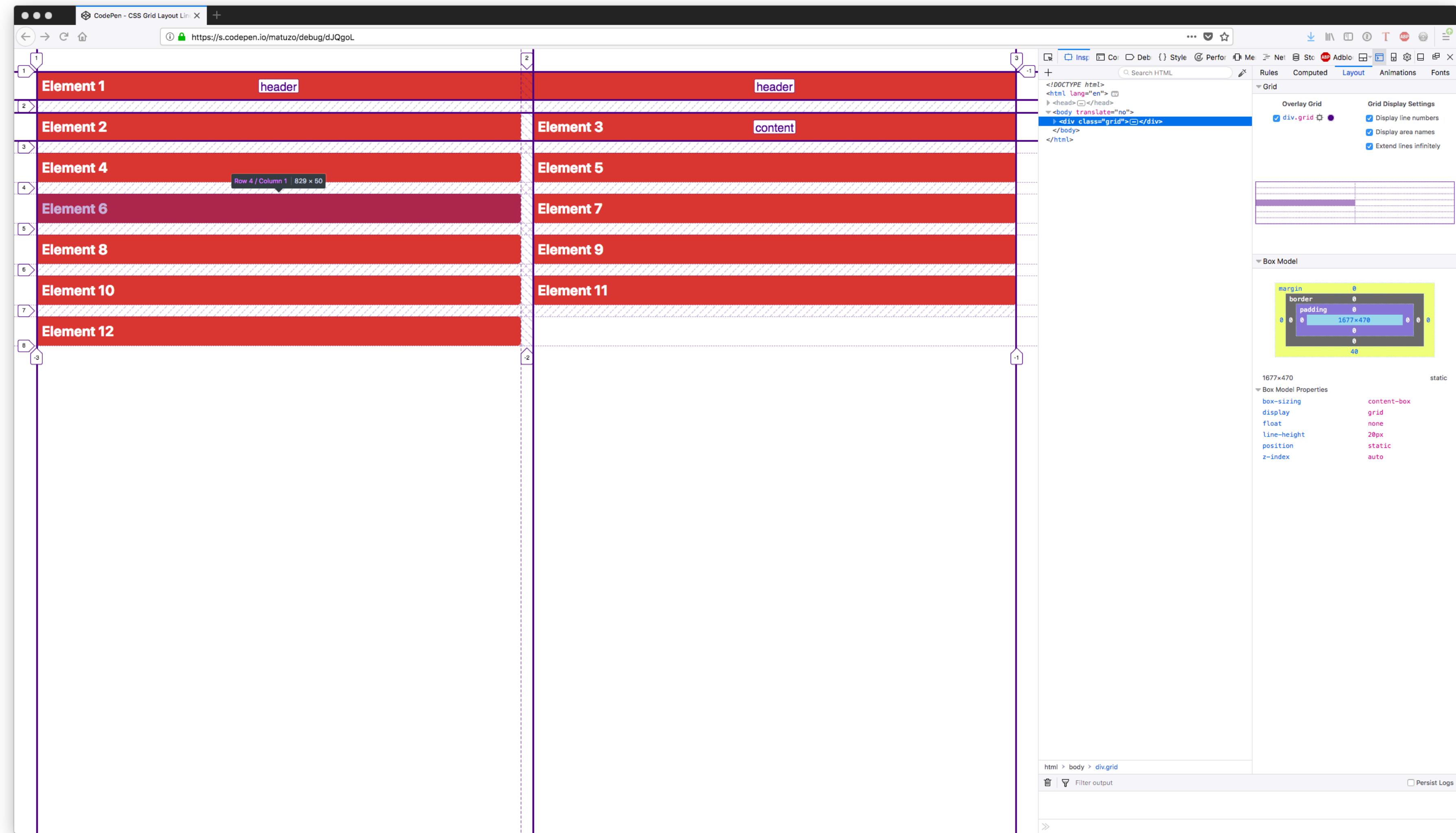
Firefox is currently the best browser for debugging.

https://developer.mozilla.org/en-US/docs/Tools/Page_Inspector/How_to/Examine_grid_layouts



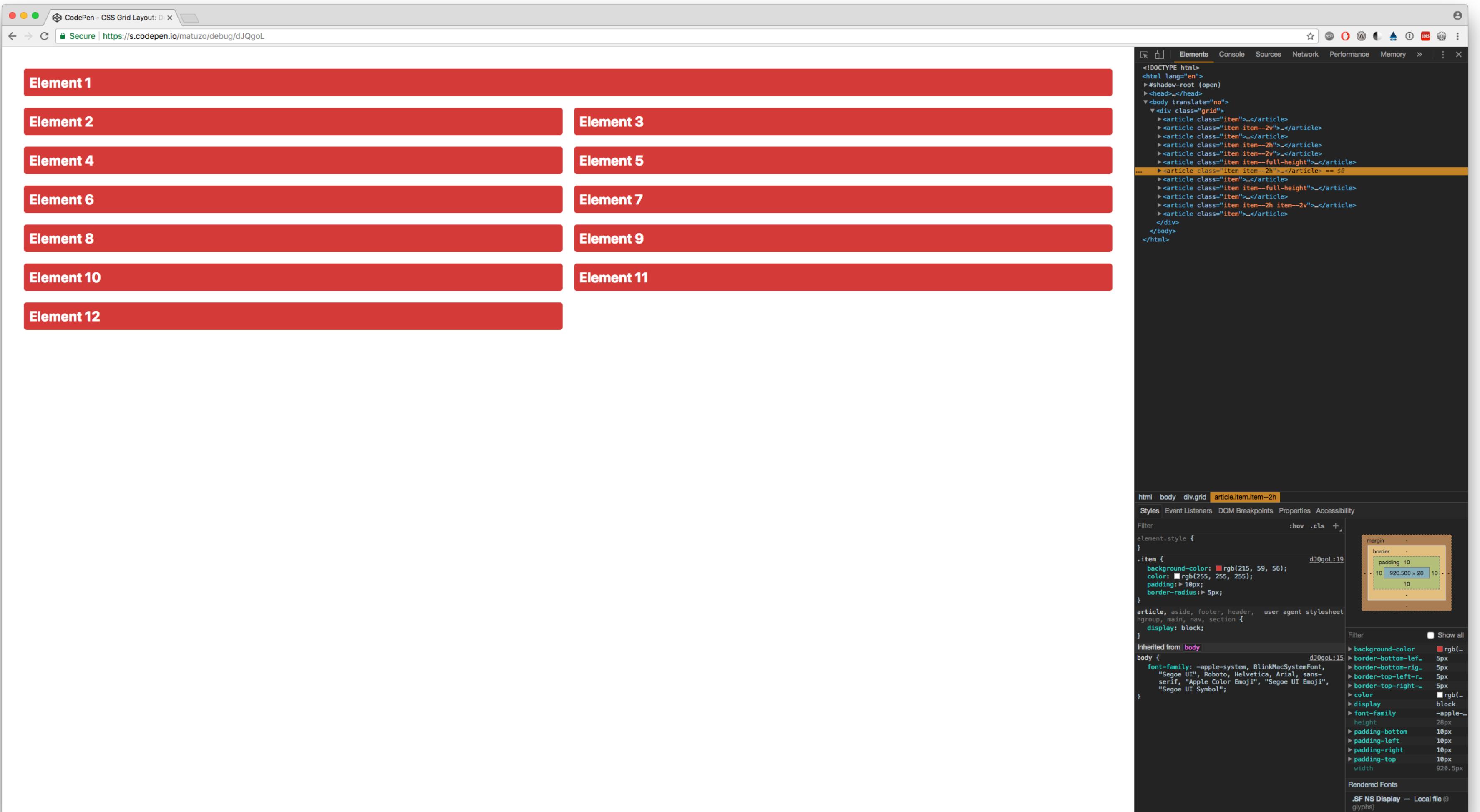
GRID DEBUGGING IN FIREFOX

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



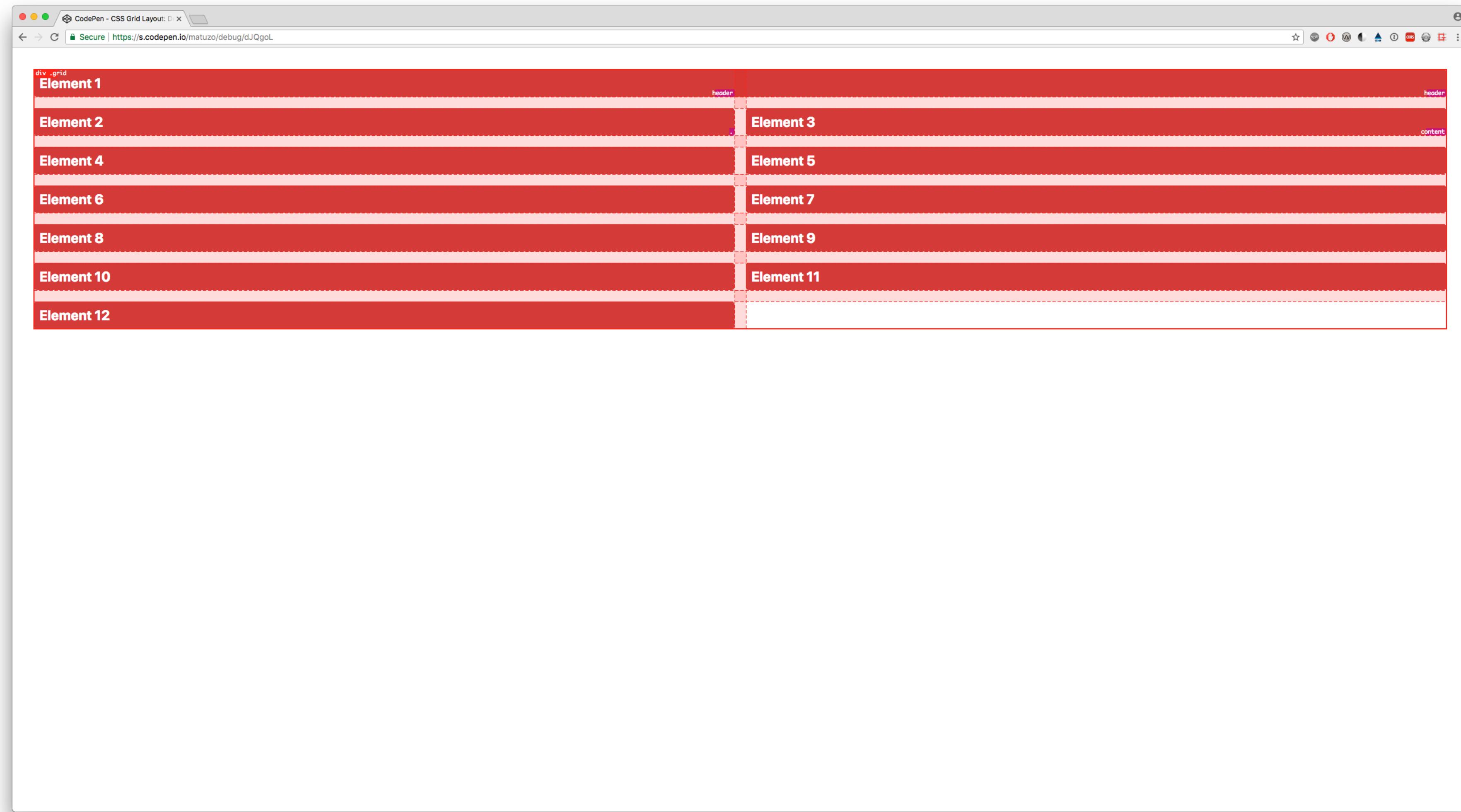
DEBUGGING IN FIREFOX

bit.ly/grid_debug



DEBUGGING IN CHROME

bit.ly/grid_debug



GRIDMAN - CSS GRID INSPECTOR. ULTRA FAST!

[Chrome](#)

GRID BASICS

fr unit, repeat, minmax

The screenshot shows a CodePen interface with a grid layout example. The grid consists of 9 elements arranged in 3 rows and 3 columns. The elements are labeled Element 1 through Element 9. The grid uses `fr` units for both columns and rows, resulting in a non-uniform layout where Element 3 is significantly larger than the others.

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: 1fr 150px 2fr;
  grid-template-rows: 1fr 50px 2fr;
  grid-gap: 20px;
}

html, body, body > * {
  height: 100%;
}

body {
  box-sizing: border-box;
  margin: 0;
```

JS

```
1
```

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 CodePen interface with three examples of CSS Grid layout using the `repeat()` function.

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>
```

```
<div class="grid">
  <article class="item">
    <h2>Element 4</h2>
  </article>
  <article class="item">
    <h2>Element 5</h2>
  </article>
  <article class="item">
    <h2>Element 6</h2>
  </article>
</div>
```

```
<div class="grid">
  <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-gap: 20px;
  /* Repeating */
}
```

```
.grid2 {
  /* Repeating a track listing */
  grid-template-columns: repeat(2, 1fr 200px);
}
```

```
.grid3 {
  /* Mixing units and lists */
  grid-template-columns: 150px repeat(3, 1fr) 200px;
}
```

JS

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

Console Assets Comments *

Save Fork Settings Change View

REPEATING TRACKS

bit.ly/grid_repeating

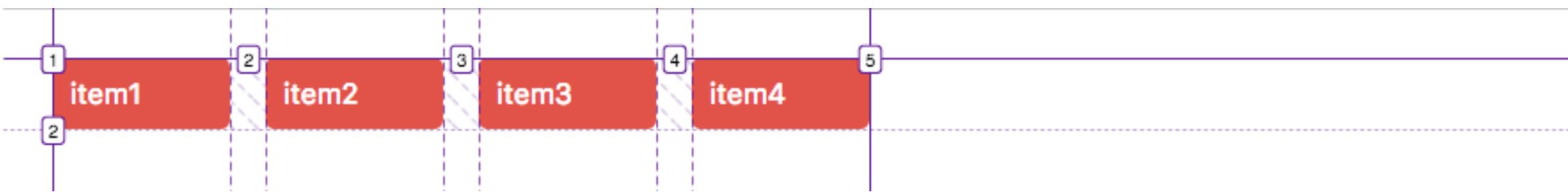
Grid Basics

Repeating tracks or track lists

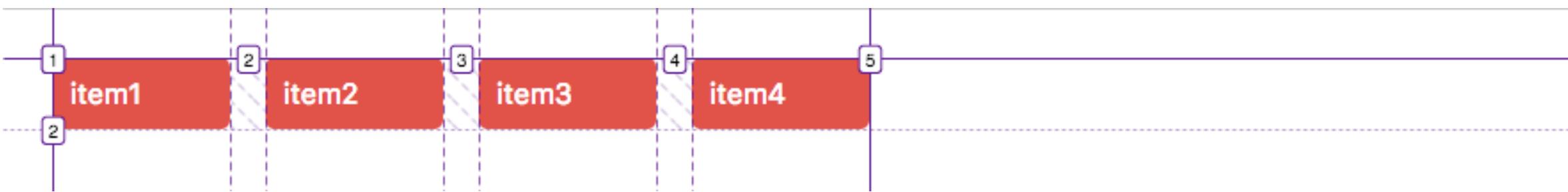
```
.container {  
    grid-template-columns: repeat(3, 200px);  
    grid-template-columns: repeat(2, 1fr 200px);  
    grid-template-columns: 150px repeat(3, 1fr) 200px;  
}
```

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

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

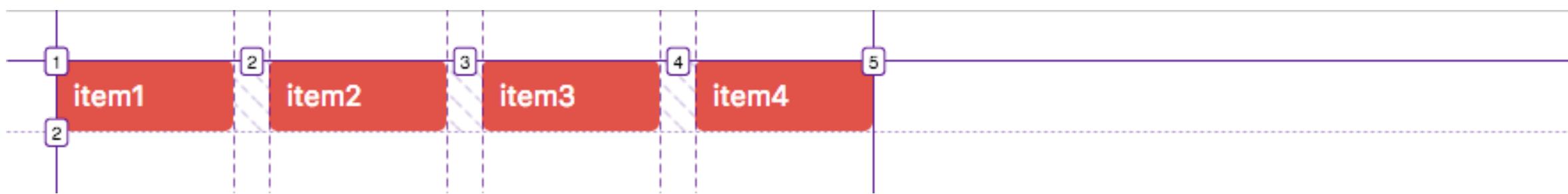


```
.grid {  
  grid-template-columns: repeat(4, 100px);  
}
```



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

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

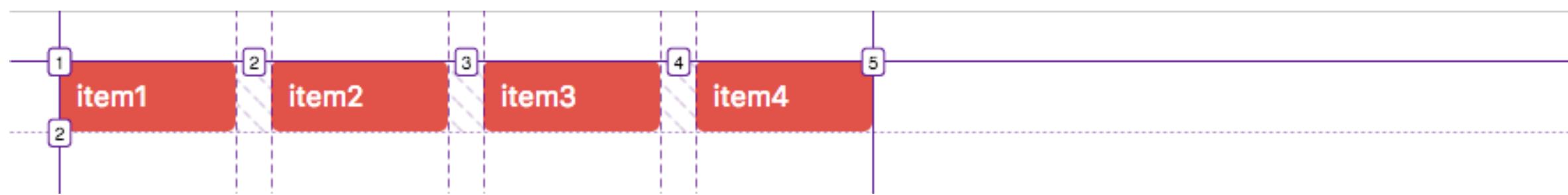


```
.grid {  
  grid-template-columns: repeat(auto-fill, 100px);  
}
```



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

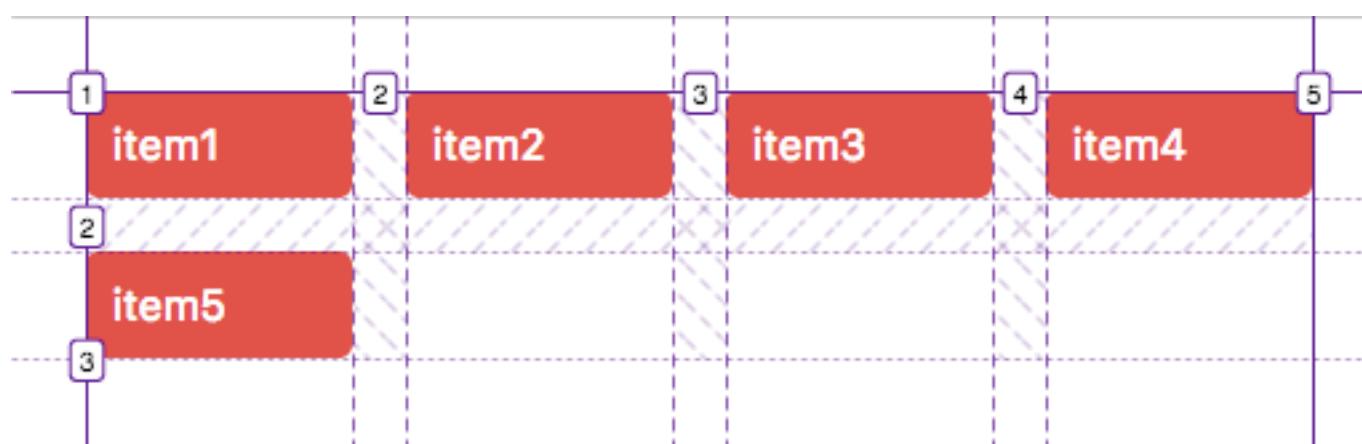
```
.grid {  
  grid-template-columns: repeat(auto-fit, 100px);  
}
```



```
.grid {  
  grid-template-columns: repeat(auto-fit, 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 CodePen interface with two examples of CSS Grid layout.

Example 1 (Top):

- HTML:**

```
<div class="grid">
  <article class="item">
    <h2>Element 1</h2>
    <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Assumenda accusantium vel accusamus, ipsa, iure reprehenderit. Totam dolore blanditiis unde velit ab ipsum necessitatibus repudiandae vero, ex praesentium mollitia deleniti adipisci?</p>
  </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>
```
- CSS:**

```
.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;
}
```

Example 2 (Bottom):

- 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>
```
- CSS:**

```
.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;
}
```

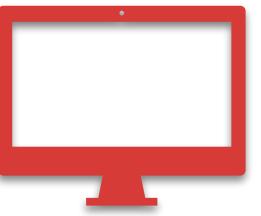
MINMAX

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);  
}
```

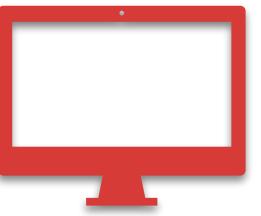


EXERCISE 2

The screenshot shows a Firefox Nightly browser window with the title "CodePen - CSS Grid Layout: im". The main content area displays a grid of 10 small images of a person wearing a blue and white patterned headscarf. The browser's developer tools are open, with the "Layout" tab selected in the Grid panel. The HTML code for the page is visible in the left pane:

```
<!DOCTYPE html>
<html>
  <head>
    </head>
  <body translate="no">
    <ul class="gallery">
      <li></li>
      <li></li>
      <li></li>
      <li></li>
      <li></li>
      <li></li>
      <li></li>
      <li></li>
      <li></li>
    </ul>
  </body>
</html>
```

The Grid panel includes settings for "Overlay Grid" and "Grid Display Settings", with "ul.gallery" selected. The "Grid Display Settings" section has three checkboxes checked: "Display line numbers", "Display area names", and "Extend lines infinitely". The "Box Model" section is also visible.



EXERCISE 2

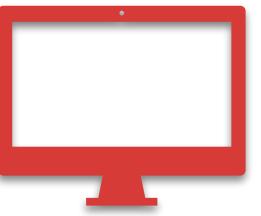
The screenshot shows a Firefox Nightly browser window with the title "CodePen - CSS Grid Layout: im". The main content area displays a grid of 10 small images of a person wearing a blue and white patterned headscarf. The browser's developer tools are open, with the "Layout" tab selected in the Grid panel. The HTML code for the page is visible in the left pane:

```
<!DOCTYPE html>
<html>
  <head>
    </head>
  <body translate="no">
    <ul class="gallery">
      </ul>
  </body>
</html>
```

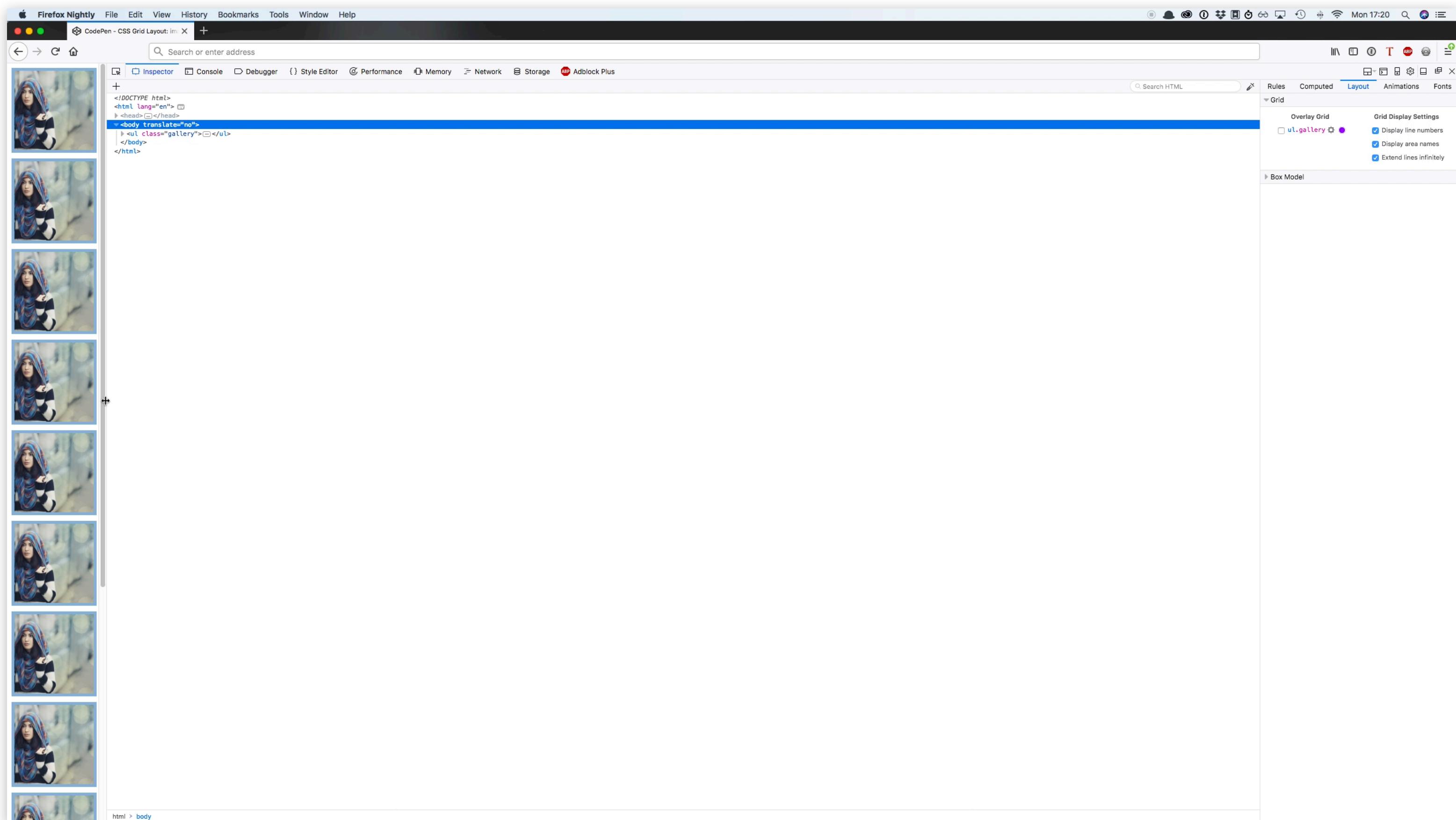
The Grid panel shows the following configuration:

- Overlay Grid:** Contains the class name `ul.gallery`.
- Grid Display Settings:** Includes checkboxes for "Display line numbers" (unchecked), "Display area names" (unchecked), and "Extend lines infinitely" (unchecked).

The bottom status bar shows the path `html > body`.



EXERCISE 2



bit.ly/grid_exercise2

CSS BOX ALIGNMENT MODULE

Box Alignment Module

Box Alignment Module

- Separate specification for alignment properties.

Box Alignment Module

- Separate specification for alignment properties.
- justify-items, align-items, justify-self, align-self, justify-content, align-content, grid-gap

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

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.



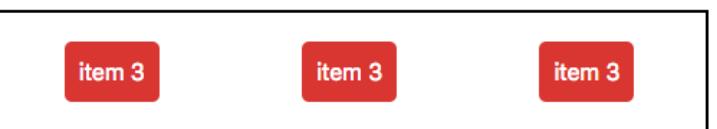
space-between

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

```

<h1>justify-content: space-evenly for flex-containers</h1>
<p><code>justify-content: space-evenly</code> <a href="https://css-tricks.com/snippets/css/complete-guide-grid/#article-header-id-20">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>
<h2>space-evenly</h2>
<p>(Works in every major desktop browser except for Edge 16)</p>
<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>
<div class="parent evenly">
  <div class="item">item 3</div>
  <div class="item">item 3</div>
  <div class="item">item 3</div>
</div>
<h2>space-between</h2>
<blockquote>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>
<div class="parent between">
  <div class="item">item 3</div>
  <div class="item">item 3</div>
  <div class="item">item 3</div>
</div>
<h2>space-around</h2>
<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.</blockquote>

```

CSS

```

body {
  max-width: 500px;
  line-height: 1.4;
}

.parent {
  display: flex;
  border: 2px solid #000000;
  margin-bottom: 30px;
  padding: 20px 0;
}

.evenly {
  justify-content: space-evenly;
}

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

```

JS

Box Alignment Module

Box Alignment Module

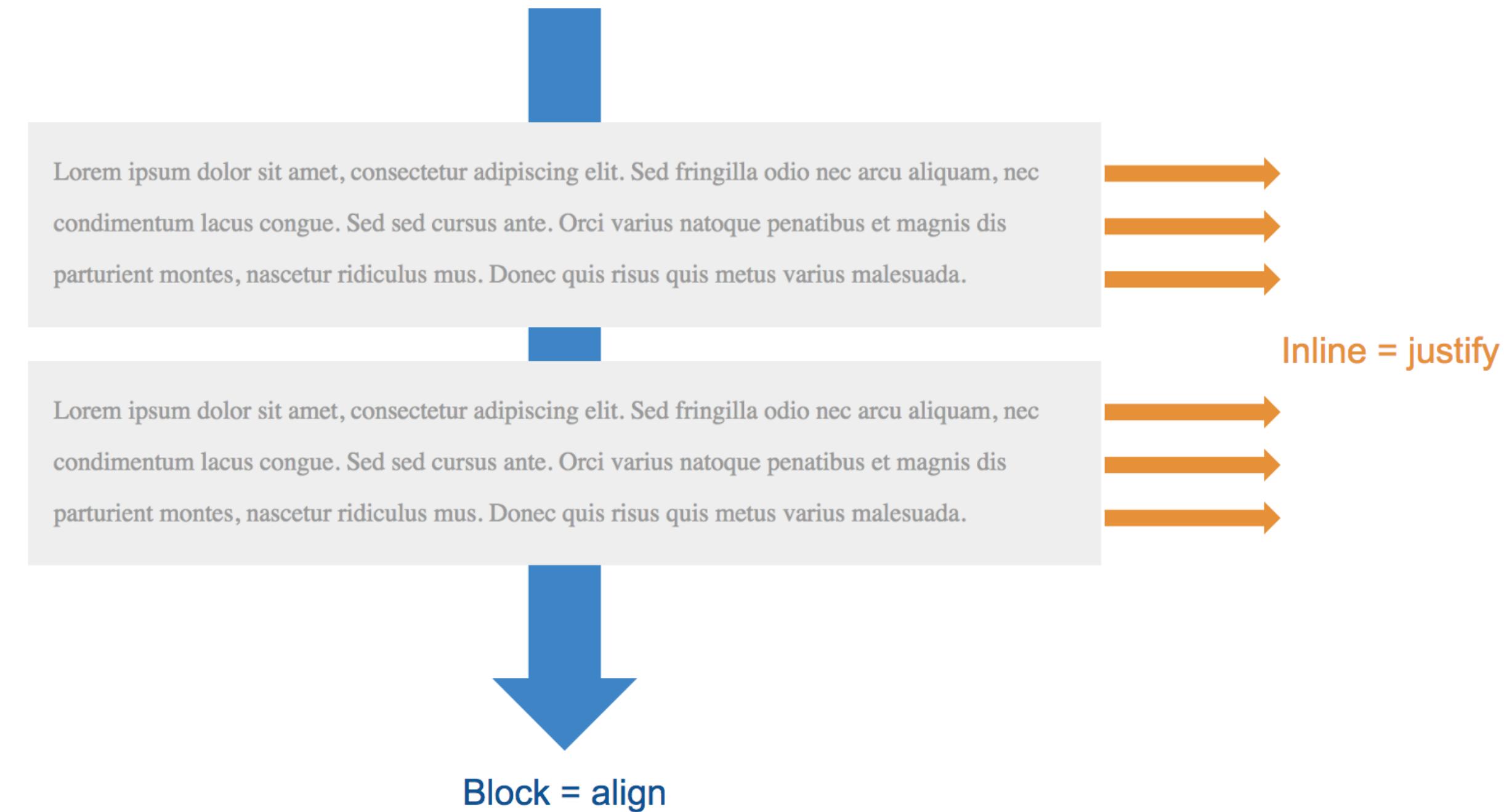
- z-axis aside, we have two axes on which we can work on.

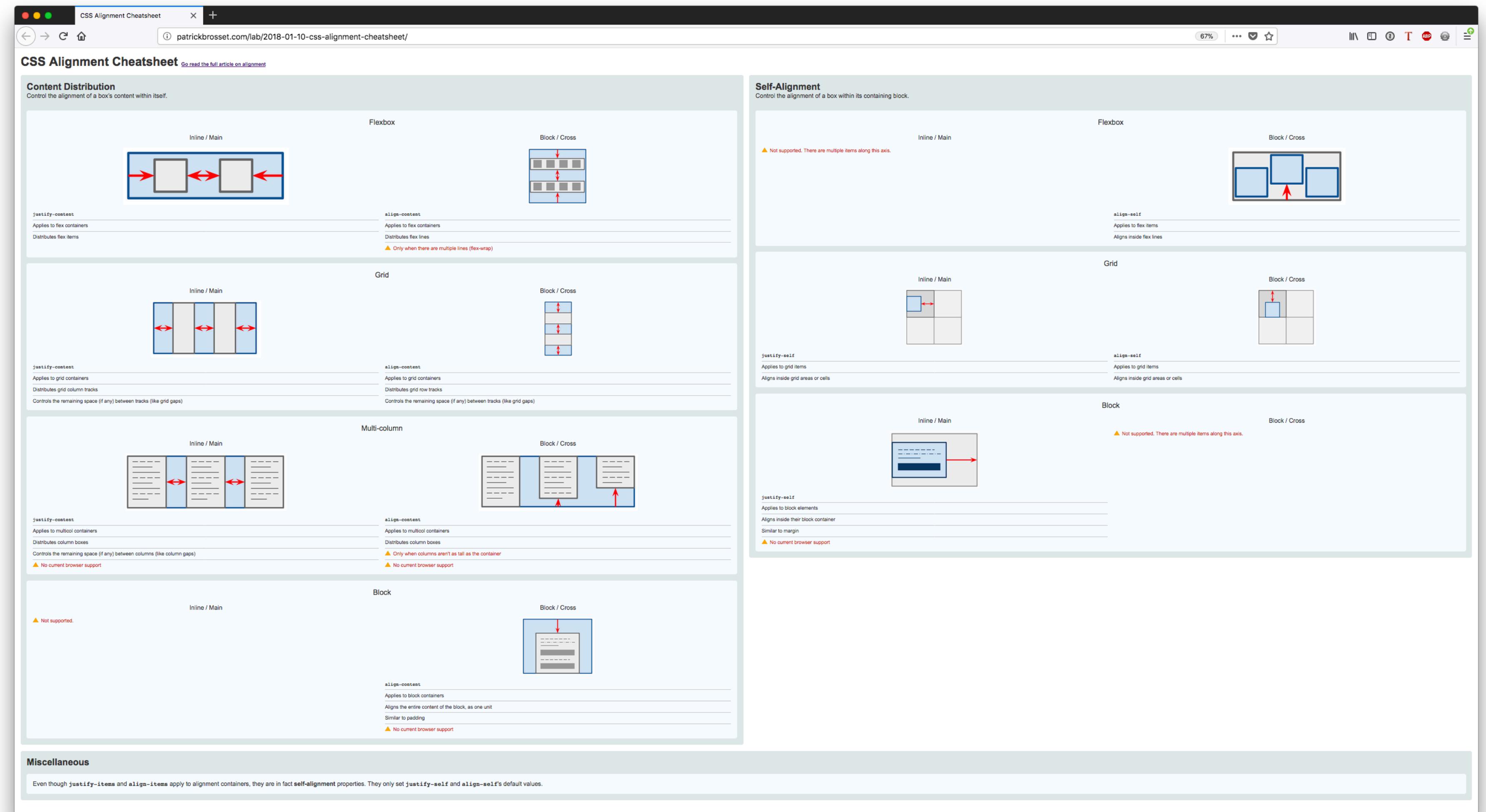
Box Alignment Module

- 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.

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

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

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

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". The main area displays four different grid layout configurations using red boxes labeled Element 1 through Element 8.

- Grid 1:** A 2x4 grid where each item is a red box with a white border. The items are centered both horizontally and vertically within their grid cells.
- Grid 2:** A 2x4 grid where each item is a red box with a white border. The items are aligned to the end of their respective grid lines.
- Grid 3:** A 2x4 grid where each item is a red box with a white border. Item 1 is aligned to the end of its row, Item 2 is centered, Item 3 is aligned to the start of its column, and Item 4 is aligned to the center of its column.
- Grid 4:** A 2x4 grid where each item is a red box with a white border. Item 5 is aligned to the end of its row, Item 6 is centered, Item 7 is aligned to the center of its row, and Item 8 is aligned to the start of its column.

The right side of the interface features three panels: "HTML", "CSS", and "JS". The "HTML" panel shows the basic structure of the eight articles. The "CSS" panel contains the following CSS code:

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

The "JS" panel is currently empty.

ALIGNING CELLS

bit.ly/grid_align

Box Alignment

Inline cell alignment

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

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

Box Alignment

Inline cell alignment

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

Block cell alignment

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

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

The screenshot shows a CodePen interface with a dark theme. On the left is a preview window displaying a grid of nine red rectangular boxes labeled Element 1 through Element 9. The grid has three columns and three rows. The right side of the interface contains the code editor with three tabs: HTML, CSS, and JS. The HTML tab shows the structure of the grid using the <grid> class and <article> elements. The CSS tab contains the following CSS code:

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

The JS tab is currently empty.

ALIGNING THE WHOLE GRID

bit.ly/grid_align2

Box Alignment

Inline grid positioning

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

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

Box Alignment

Inline grid positioning

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

Block positioning

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

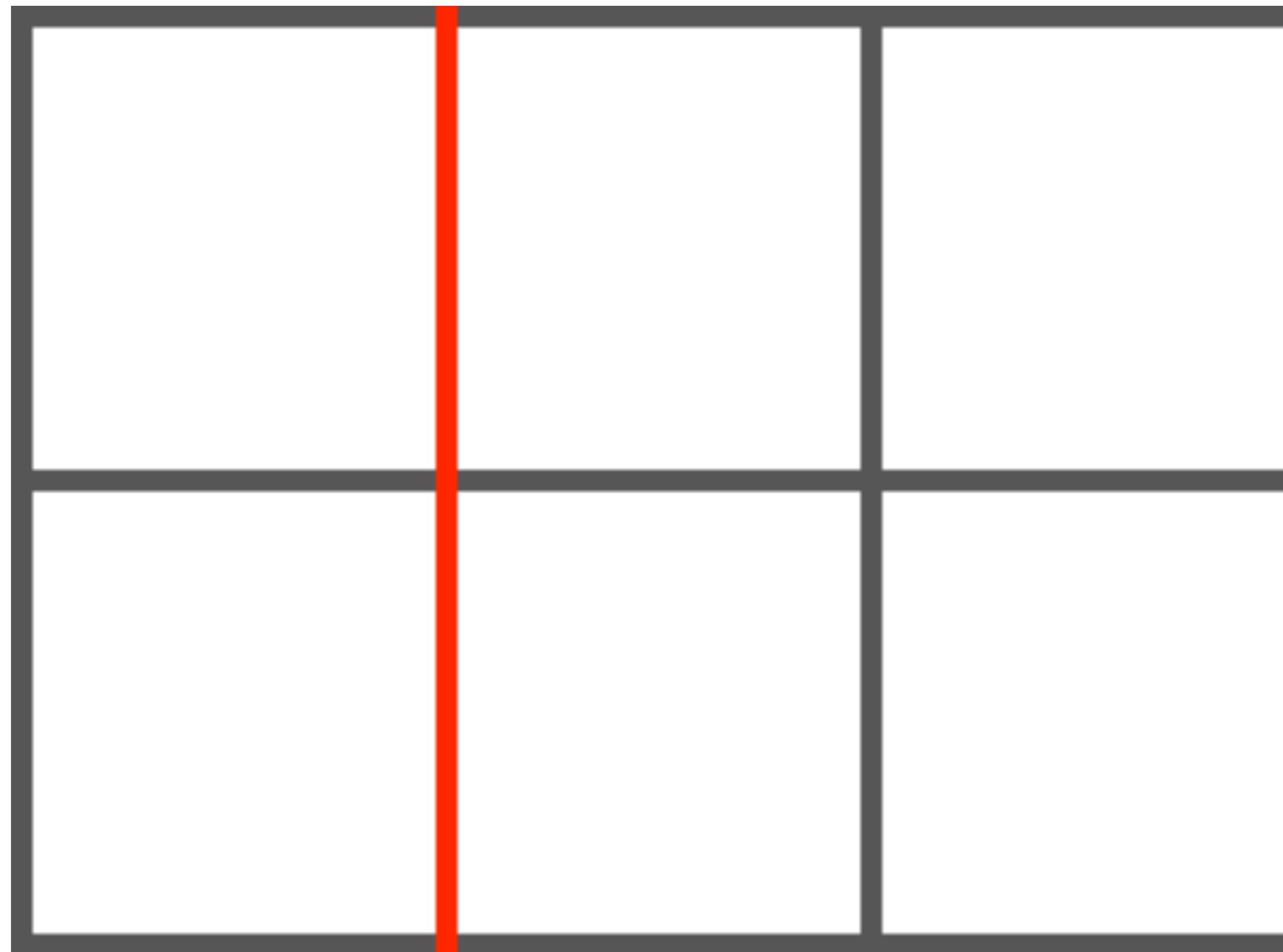
Values: **start** (default), **end**,
center, **space-between**,
space-around, **space-evenly**



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

Terminology

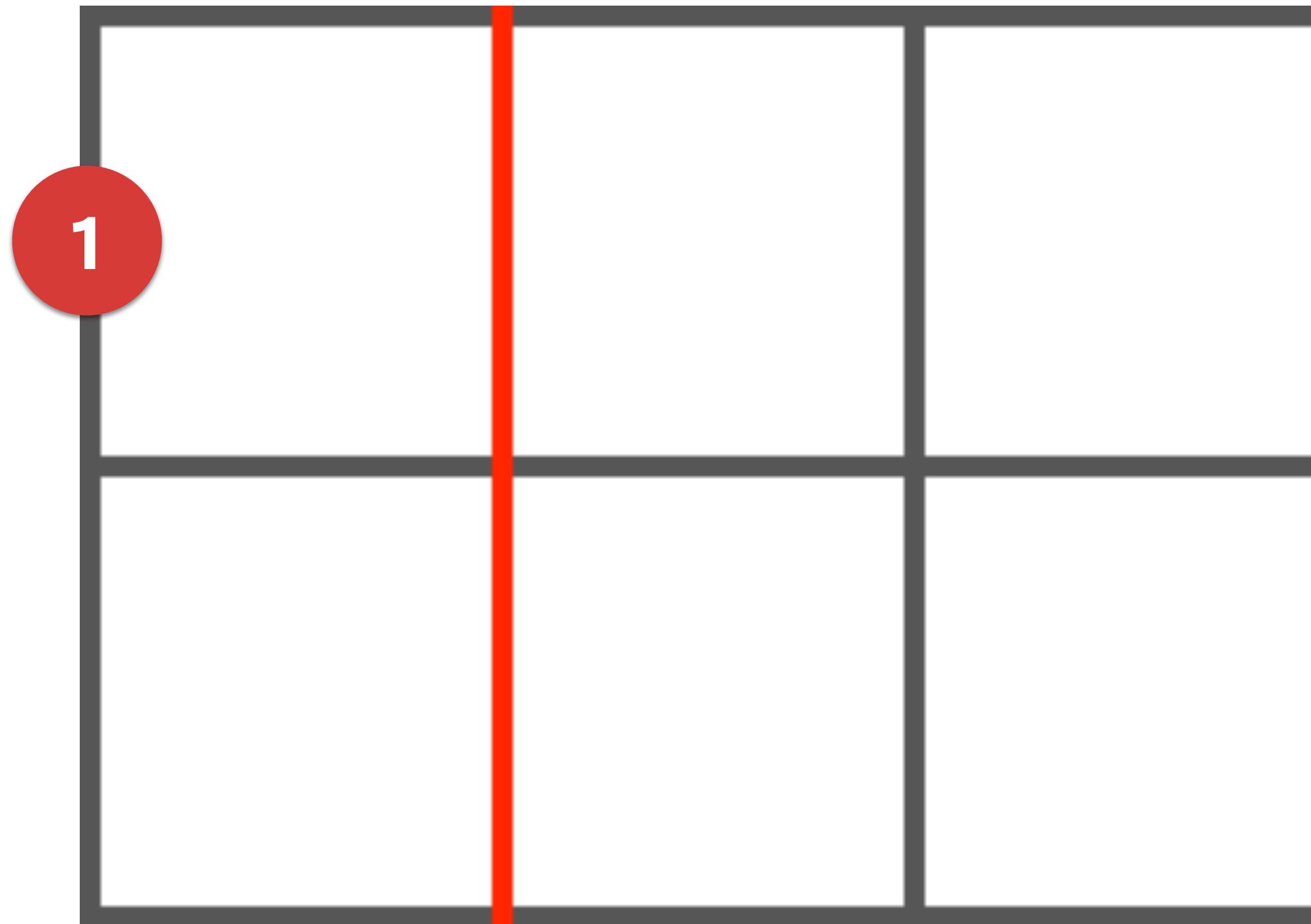
LINE



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

Terminology

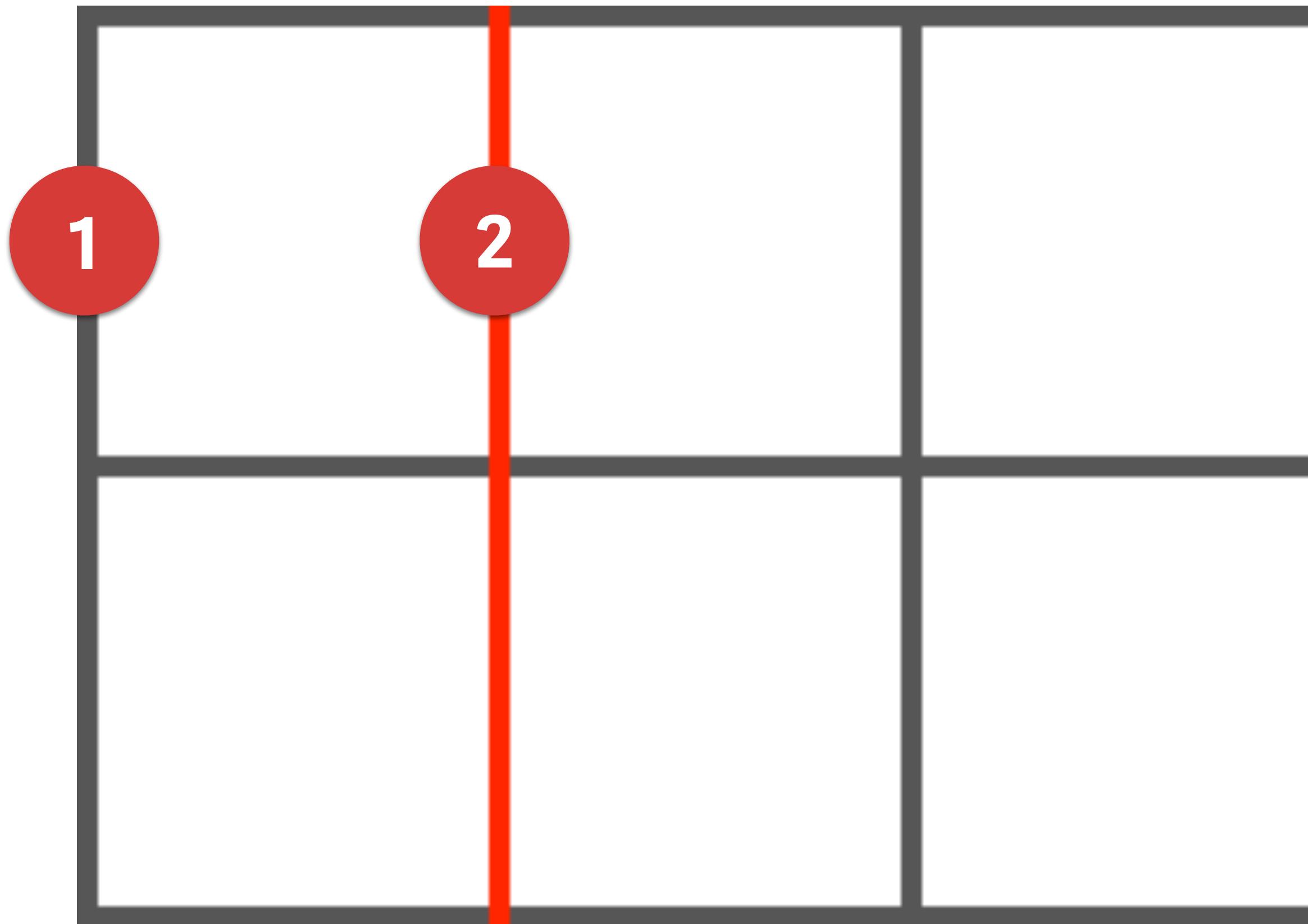
LINE



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

Terminology

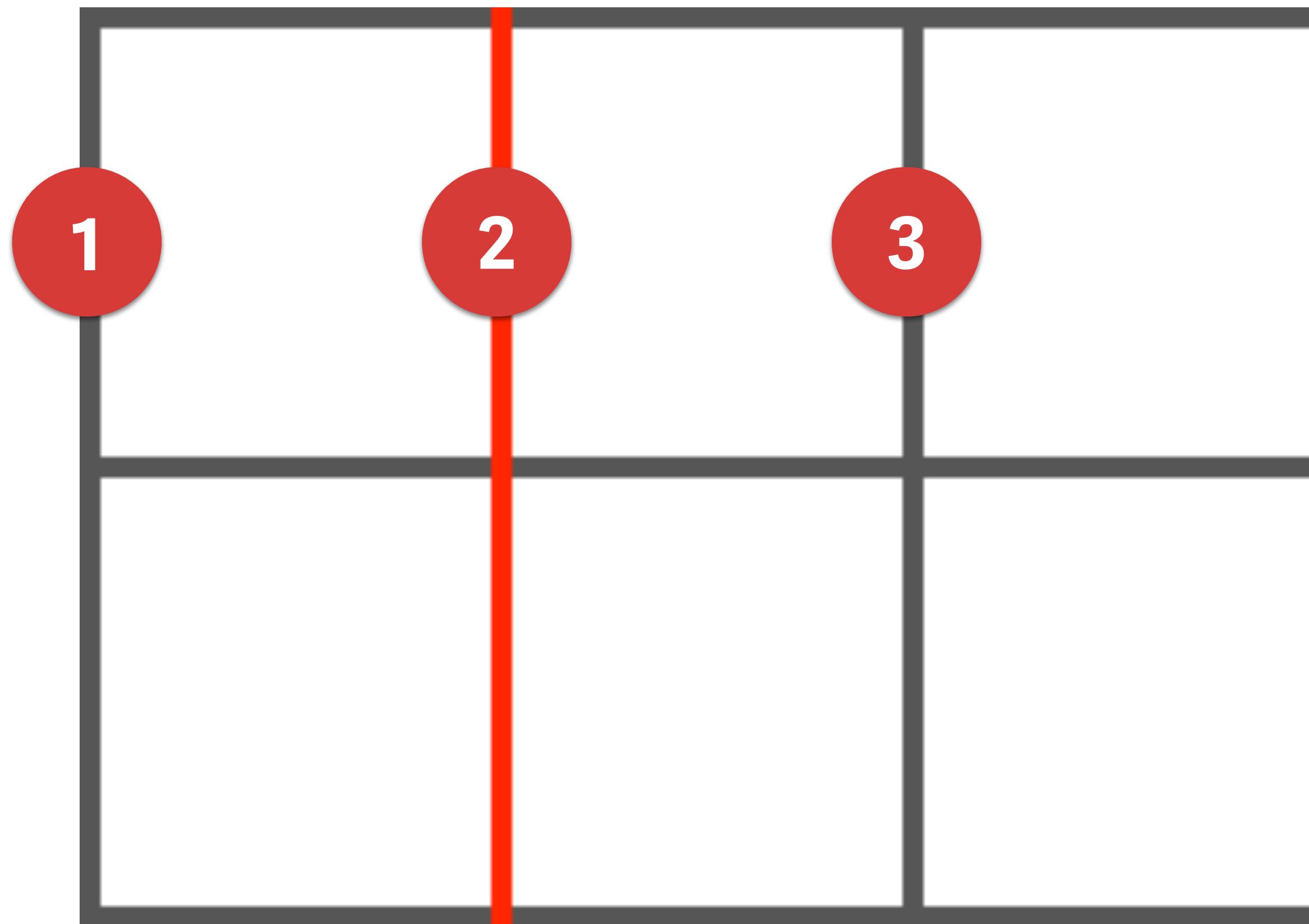
LINE



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

Terminology

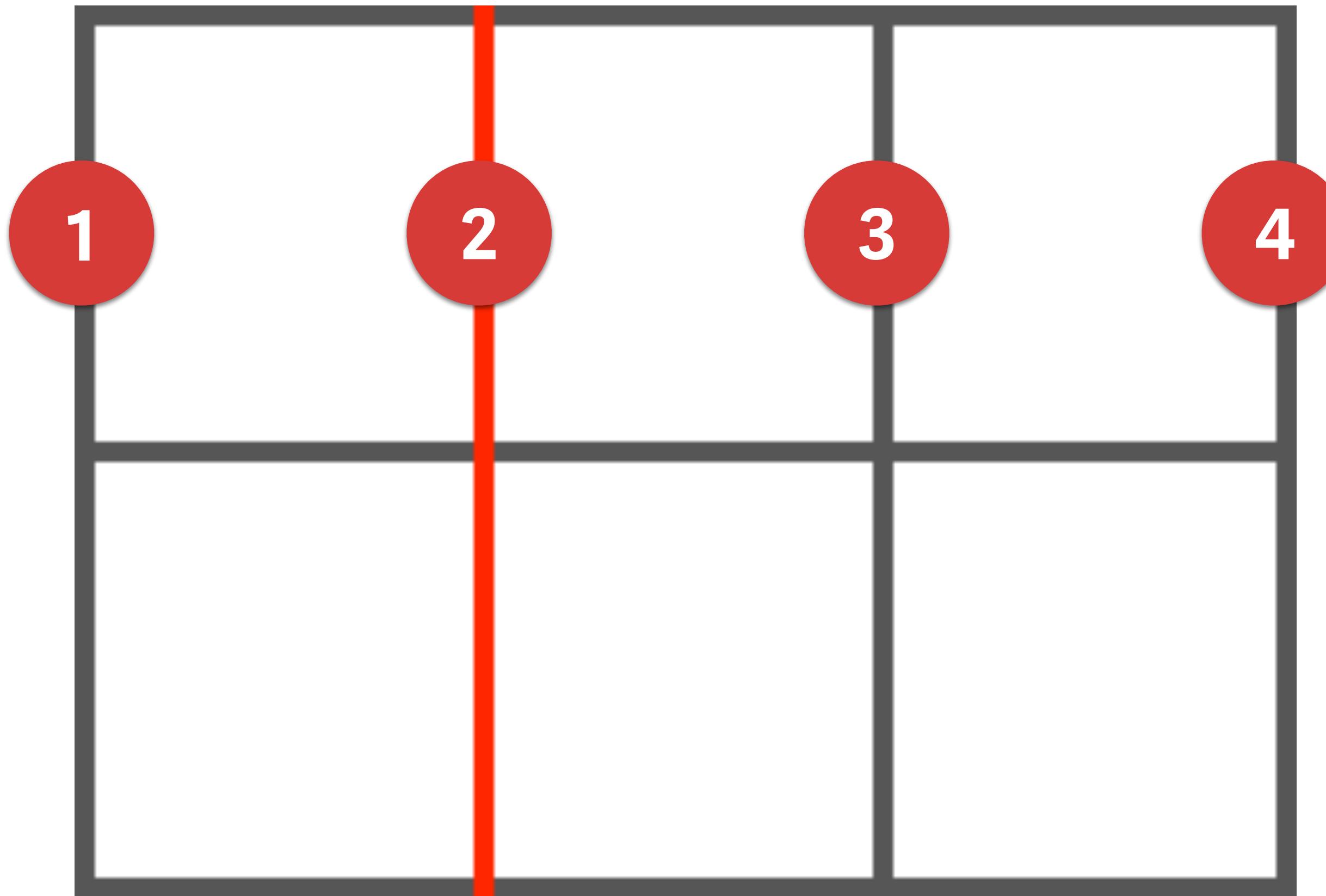
LINE



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

Terminology

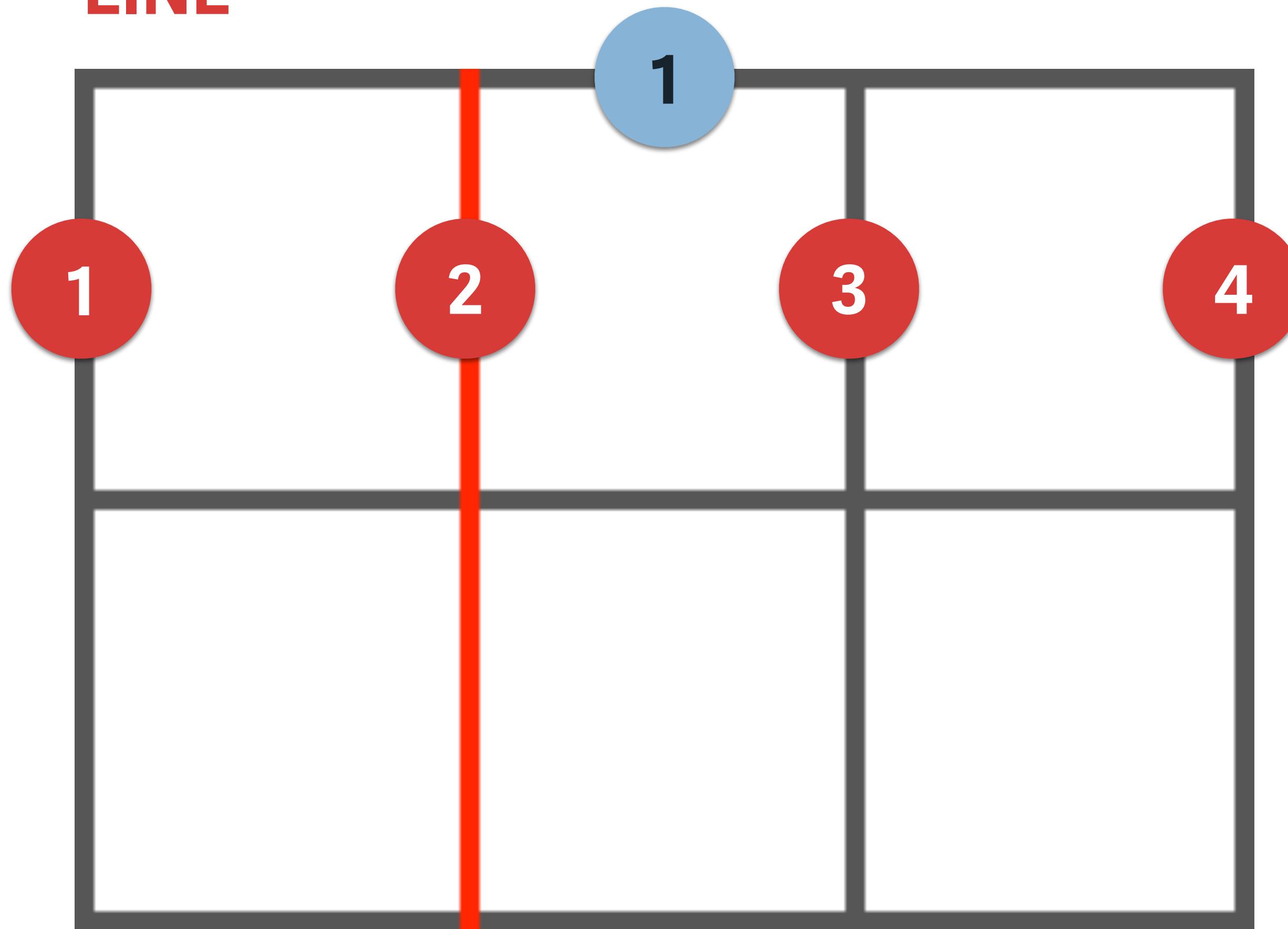
LINE



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

Terminology

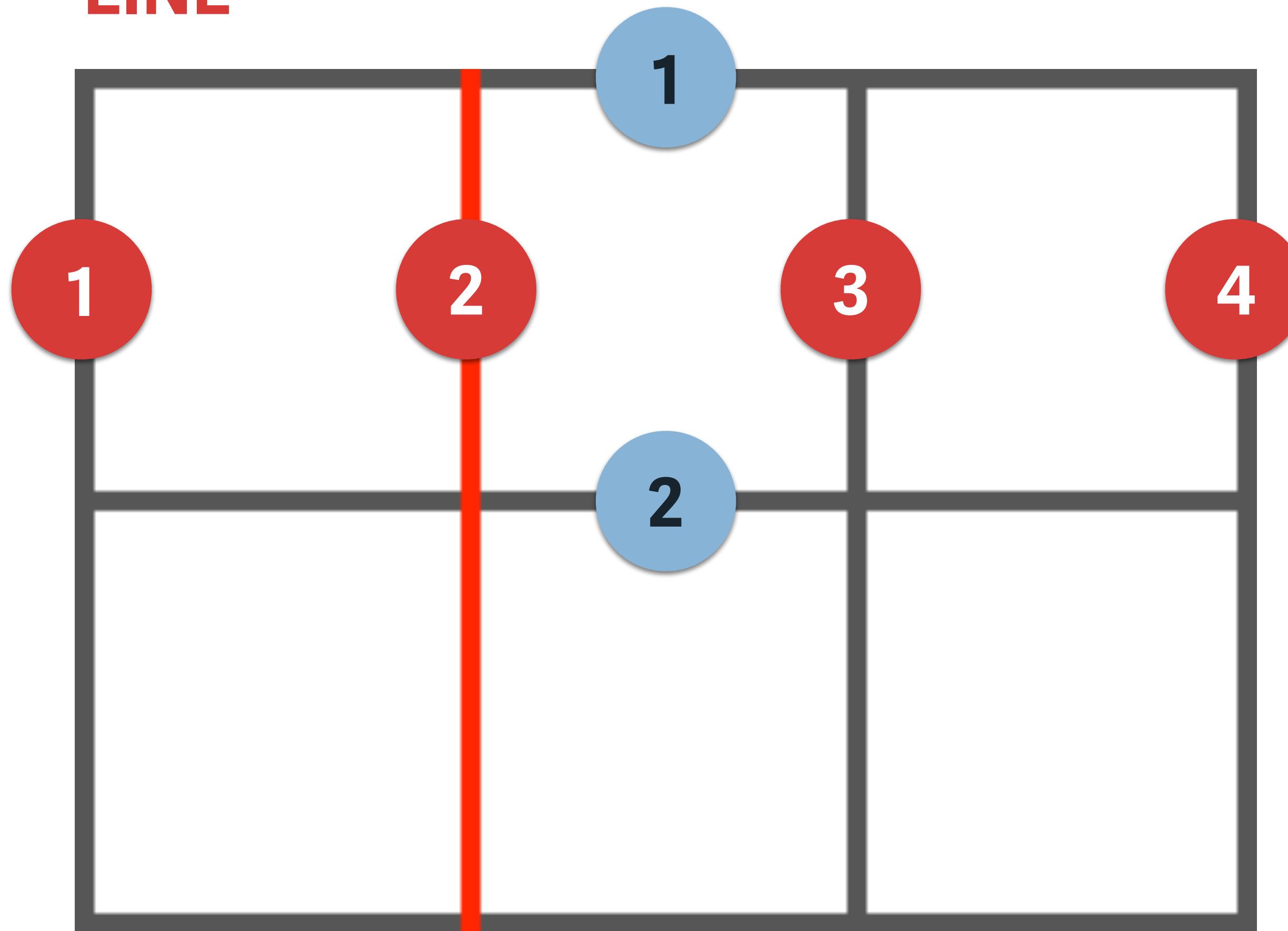
LINE



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

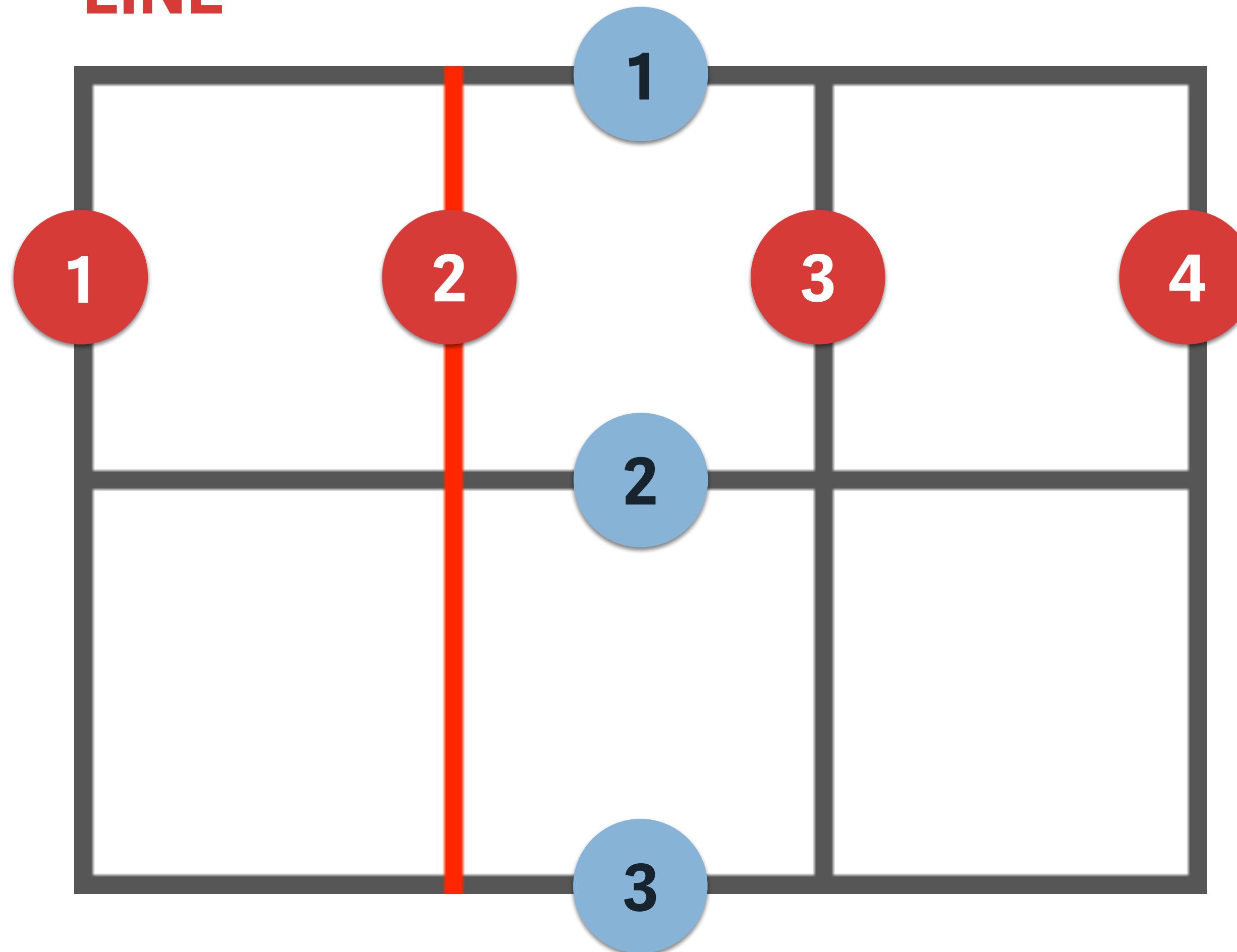
Terminology

LINE



Terminology

LINE



The screenshot shows a CodePen interface with a dark theme. On the left is a preview window displaying a grid of twelve red rectangular elements labeled Element 1 through Element 12. The grid has four columns and three rows. Elements 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, and 12 are arranged in a specific pattern across the grid. To the right of the preview are three tabs: 'HTML', 'CSS (SCSS)', and 'JS'. The 'HTML' tab shows the structure of the grid using the 'grid-template-columns' and 'grid-auto-rows' properties. The 'CSS (SCSS)' tab contains the SCSS code for defining the grid and applying column and row spans to individual items. The 'JS' tab is currently empty.

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

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

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

.item:nth-child(5) {
  grid-column-end: 4;
}

.item:nth-child(7) {
  grid-column-start: 1;
  grid-column-end: -1;
}

.item:nth-child(11) {
  grid-column-start: 2;
  grid-row-start: 4;
}

.item:nth-child(9) {
  grid-column: 3 / 5;
}

.item:nth-child(8) {
  grid-row-start: 6;
  grid-row-end: 8;
}

.item:nth-child(10) {
  grid-column: 2 / span 2;
  grid-row: span 4;
}
```

PLACING CELLS ON LINES

bit.ly/grid_lines

Lines

Cell start and end point.
(vertical track)

Value: Line number

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

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

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

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 grid layout with two items: "item1" and "item2". The grid has four columns defined by the CSS rule ".grid { grid-template-columns: 1fr 1fr 1fr 1fr; }". The first item spans from column 2 to column 4, while the second item spans from column 3 to column 4. The CSS code includes a SCSS rule for the first item that sets its grid-column-end value to 2 and its grid-column-start value to 4.

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

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

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

LOWER END VALUE THAN START VALUE

bit.ly/grid_lines7

Negative lines values

Negative lines values

- Each line is represented by a positive and a negative number.

Negative lines values

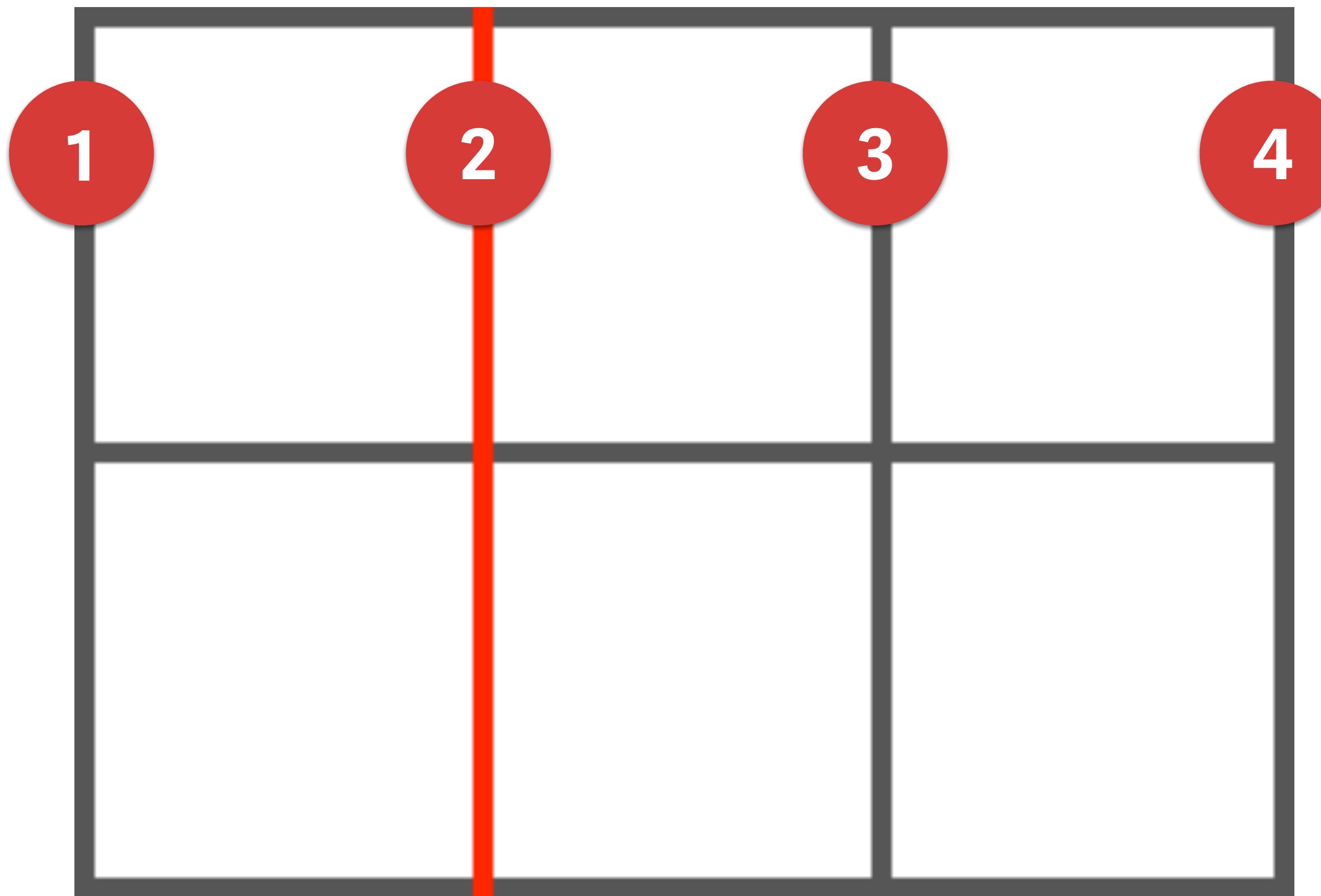
- Each line is represented by a positive and a negative number.
- For example, grid-column: 2 / -1; spans from the second line to the last line.

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

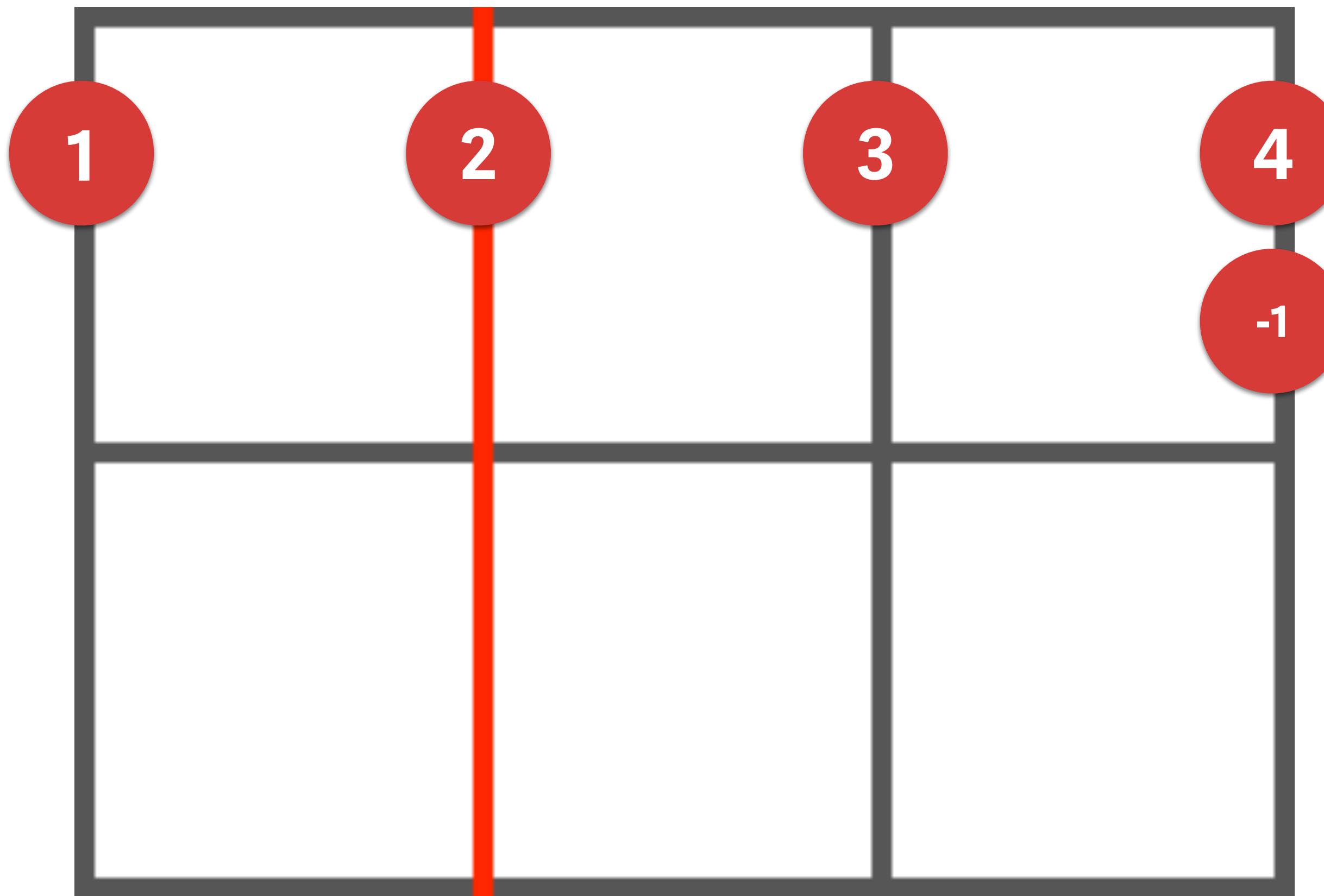
LINE



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

Terminology

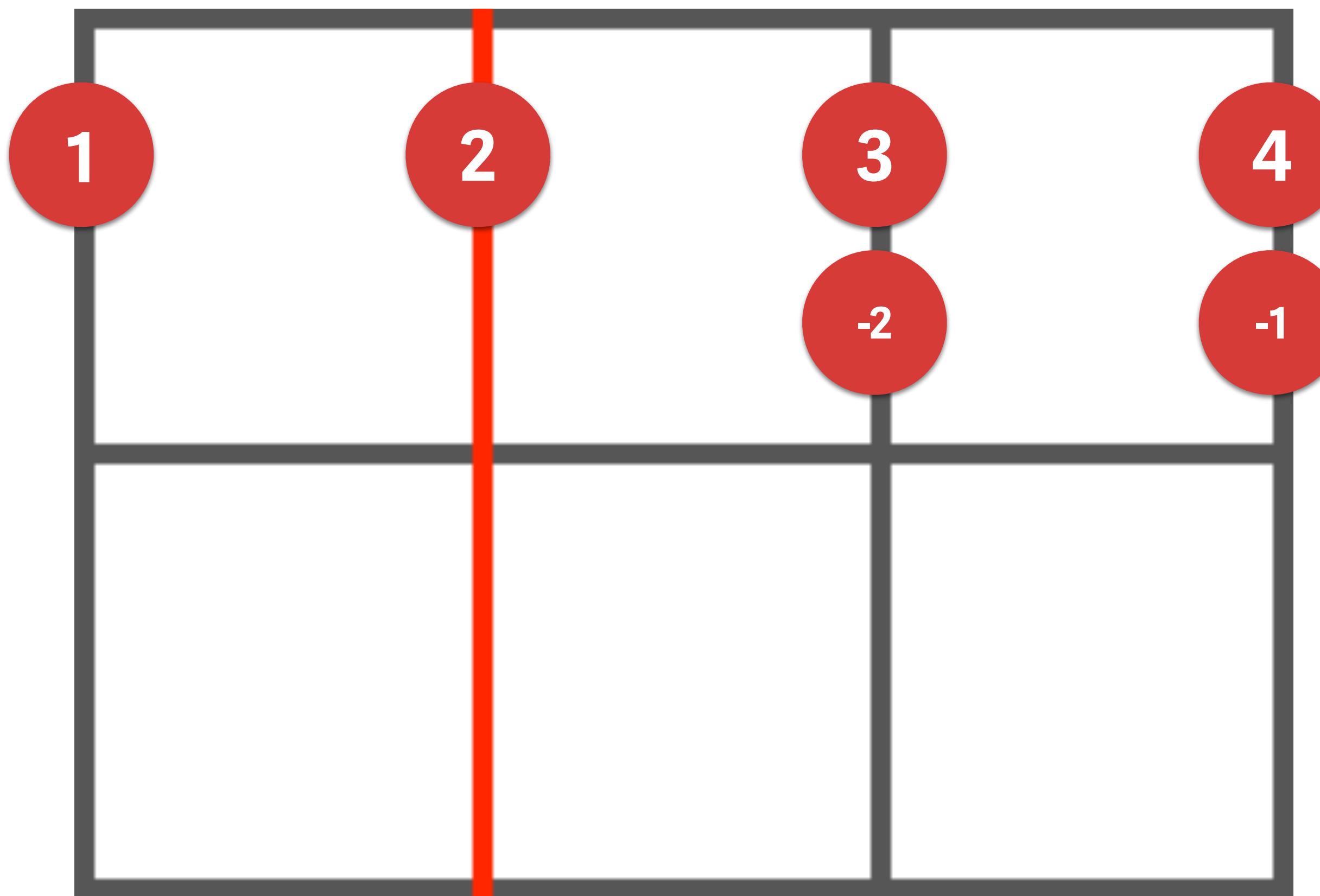
LINE



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

Terminology

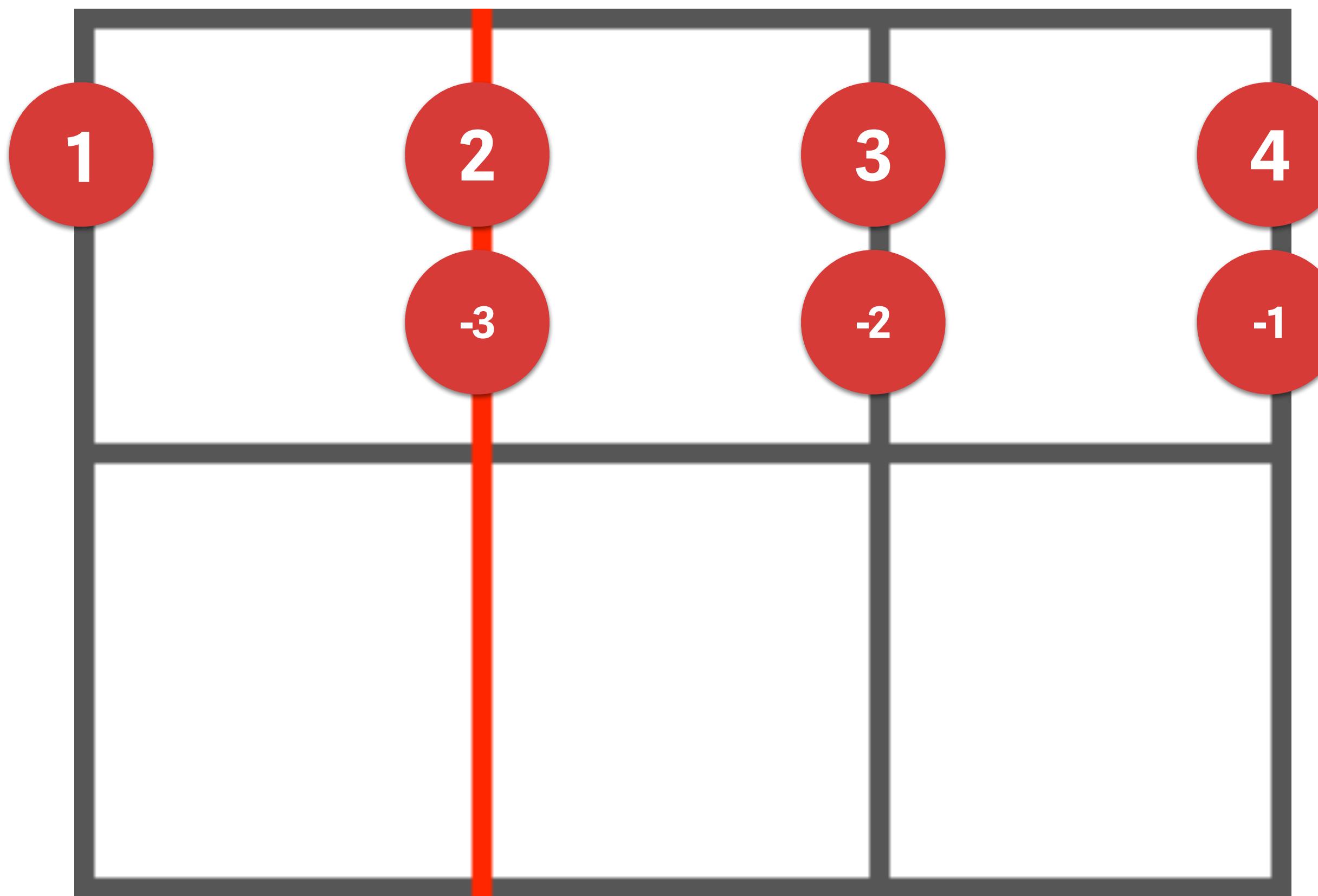
LINE



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

Terminology

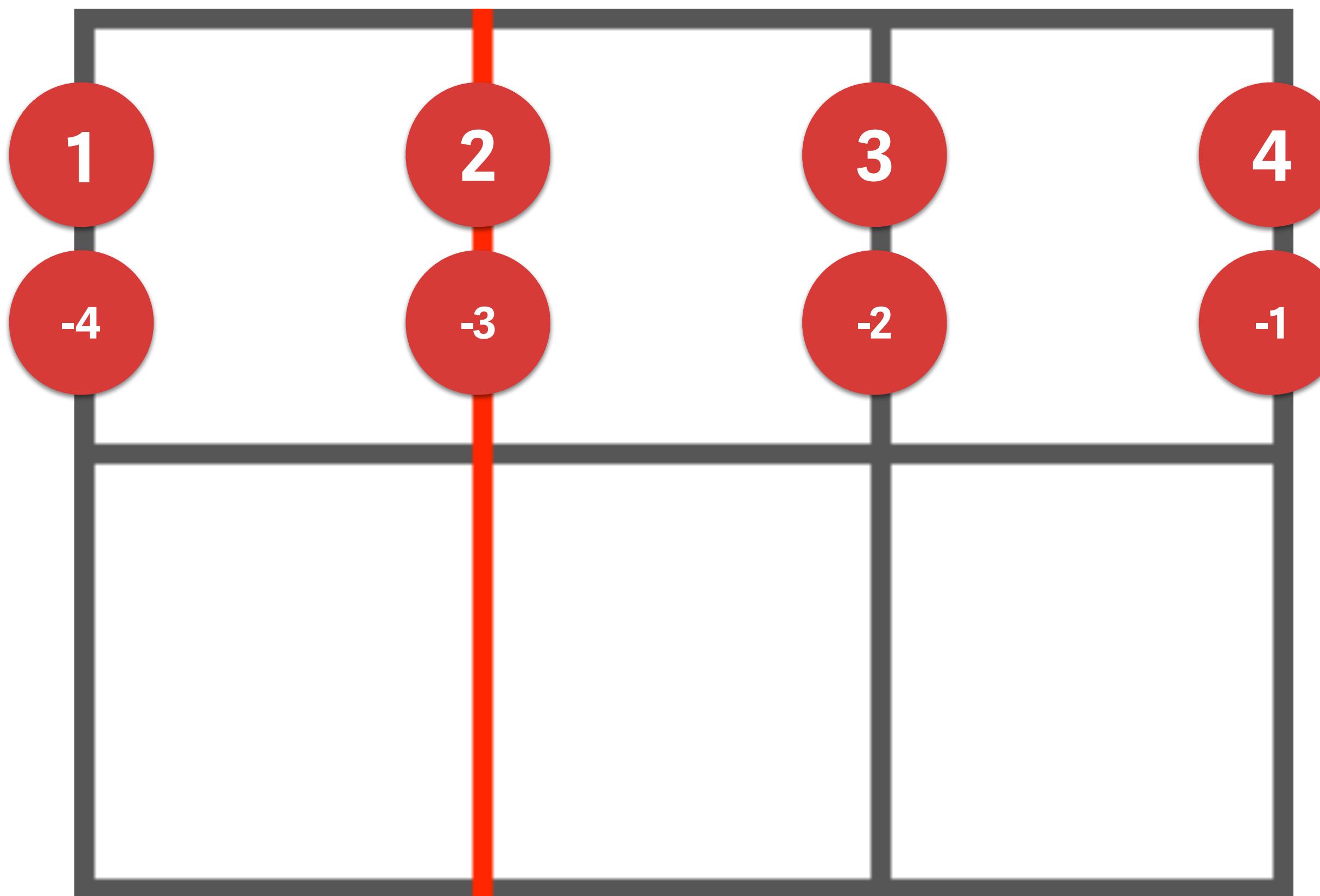
LINE



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

Terminology

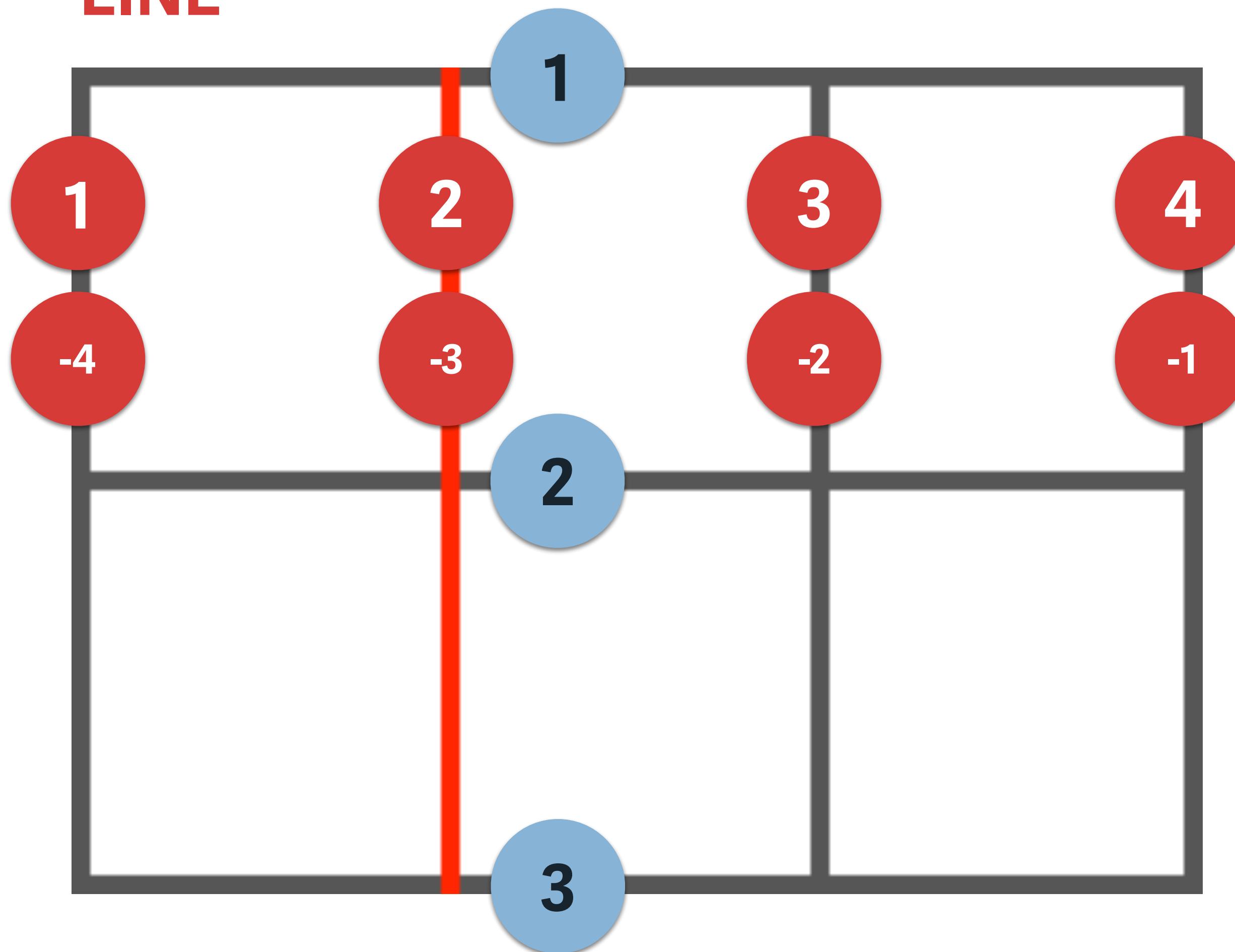
LINE



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

Terminology

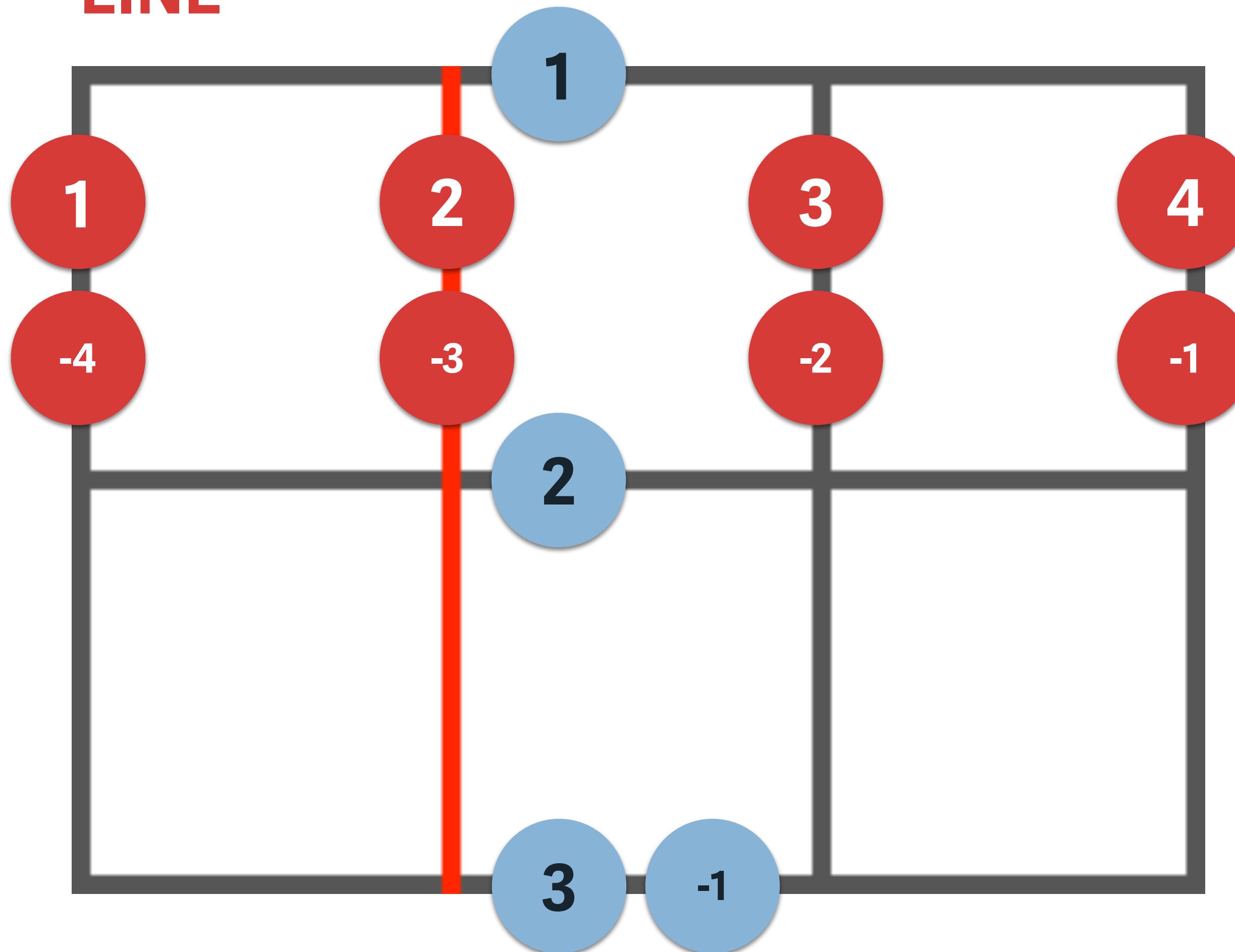
LINE



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

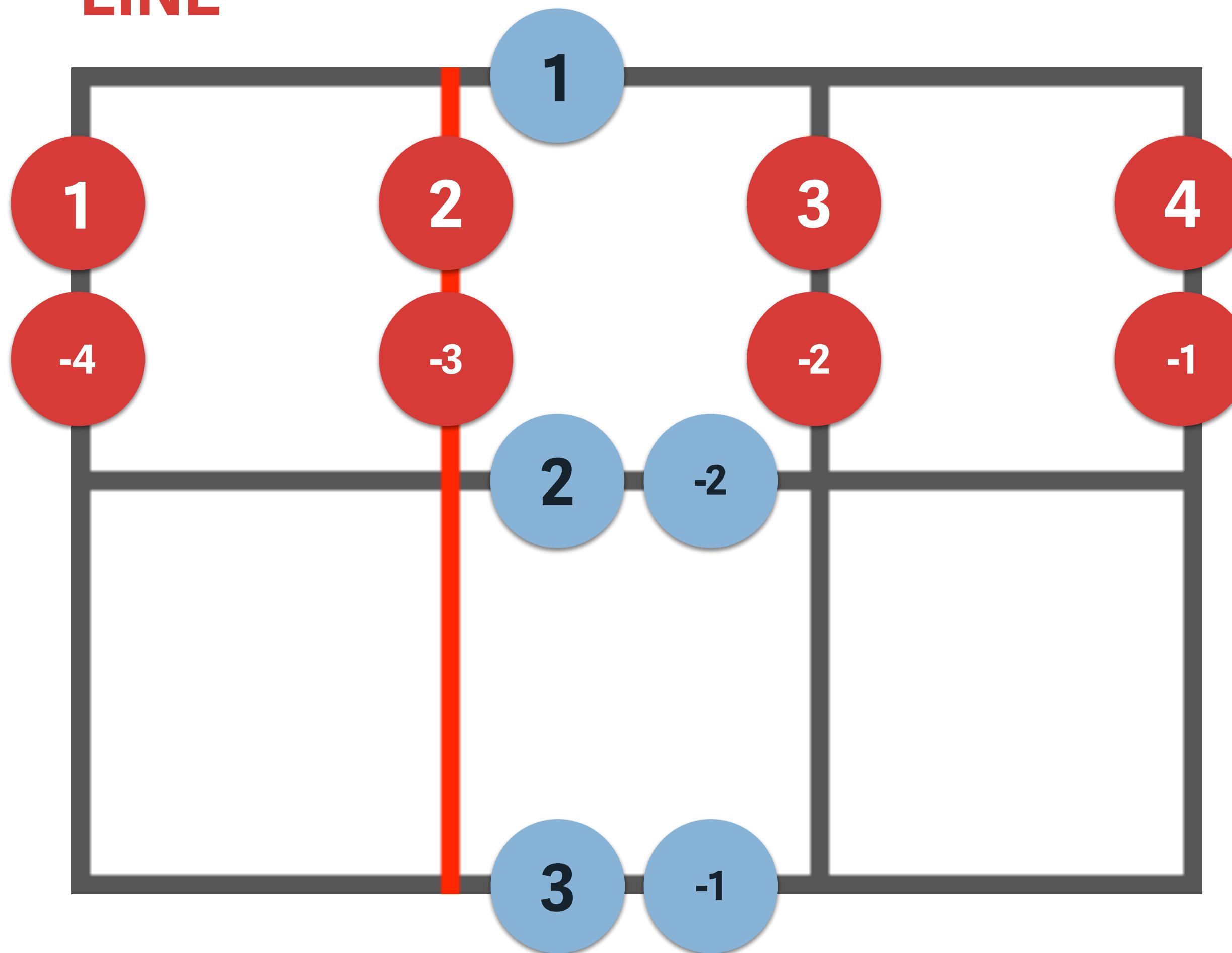
Terminology

LINE



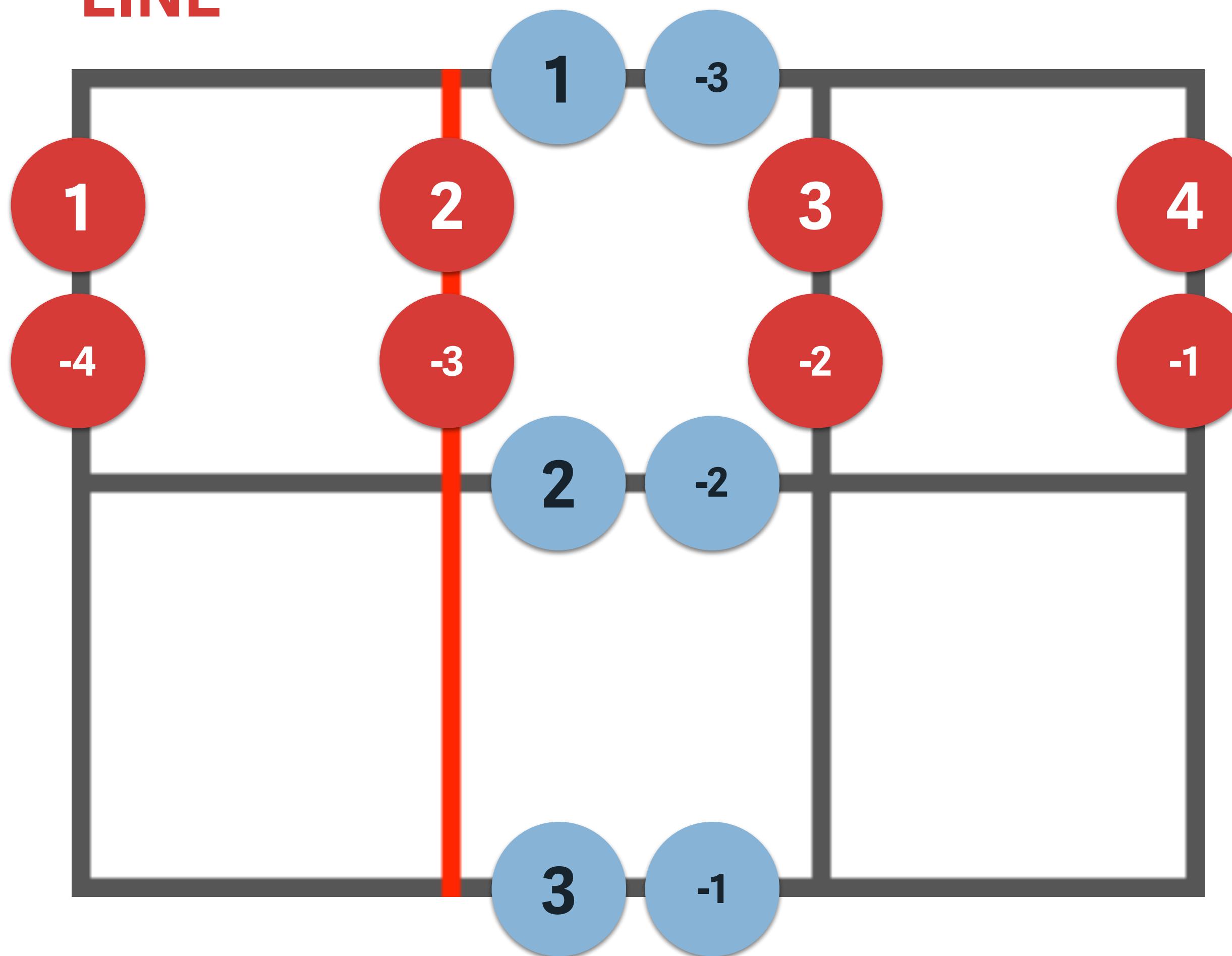
Terminology

LINE



Terminology

LINE



A screenshot of a CodePen editor window titled "Grid item from first to second to last". The page URL is <https://codepen.io/matuzo/pen/rmQMyp?editors=1100>. The editor interface includes tabs for HTML, CSS, and JS, and sections for Save, Fork, Settings, and Change View.

The HTML code 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>
</div>
```

The CSS code is:

```
.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;
}
```

NEGATIVE LINE VALUES

bit.ly/grid_lines5

A screenshot of a CodePen editor window titled "Negative values in grid-column/row-start". The window shows a grid layout with three red rectangular elements labeled "Element 1", "Element 2", and "Element 3". The elements are arranged in a 3x1 grid. Element 1 is at the bottom-left, Element 2 is at the top-right, and Element 3 is at the top-left.

The HTML code 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>
</div>
```

The CSS (SCSS) code is:

```
.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;
}
```

NEGATIVE LINE VALUES

bit.ly/grid_lines6

span Keyword

span Keyword

- A grid item spans a single cell by default.

span Keyword

- A grid item spans a single cell by default.
- That can be changed by applying the **span** keyword.

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.

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.

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 interface with the title "CSS Grid Layout: span keyword". The page displays a grid layout with three items: "item1" (a large red box), "item2" (a medium red box), and "item2" (another medium red box). The CSS code uses the `grid-column` and `grid-column-start` properties with the `span` keyword to achieve this layout.

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

```
CSS (SCSS)
.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;  
}
```

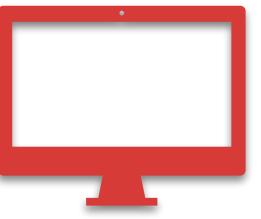
Lines

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

The screenshot shows a CodePen interface with a dark theme. The main area displays a grid of 12 red rectangular elements labeled Element 1 through Element 12. The grid uses a 4-column template with varying row heights. The CSS code on the right demonstrates how to achieve this layout using grid-auto-flow: dense;.

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

```
CSS
.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 page titled "My Website" with three articles and a sidebar. The layout uses named grid lines to organize the content.

HTML:

```
<div class="grid">
  <div class="header"></div>
  <div class="posts">
    <div class="post">Article 1</div>
    <div class="post">Article 2</div>
    <div class="post">Article 3</div>
  </div>
  <div class="partners">
    <div class="partner">Ad 1</div>
    <div class="partner">Ad 2</div>
  </div>
  <div class="aside">
    <h2>Related Posts</h2>
    <ul>
      <li>Element 1</li>
      <li>Element 2</li>
      <li>Element 3</li>
      <li>Element 4</li>
      <li>Element 5</li>
      <li>Element 6</li>
    </ul>
    <h2>More Posts</h2>
    <ul>
      <li>Element 1</li>
      <li>Element 2</li>
      <li>Element 3</li>
      <li>Element 4</li>
      <li>Element 5</li>
      <li>Element 6</li>
    </ul>
  </div>
</div>
```

CSS:

```
.grid {
  display: grid;
  grid-template-columns: 1fr 200px 200px;
  grid-gap: 20px;
  grid-auto-flow: dense;
  align-items: start;
}

.header {
  background-color: #17242D;
  grid-column-start: 1;
  grid-column-end: -1;
}

.article {
  grid-column-start: 1;
}

.posts {
  grid-column-start: 3;
  grid-row-end: span 3;
}

.partners {
  grid-column-start: 2;
}

.post {
  background-color: #87B3D6;
  margin-bottom: 20px;
}

.partner {
  background-color: #7CCBB0;
  margin-bottom: 20px;
}
```

JS:

```
/* No JavaScript code present */
```

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;  
}
```

A screenshot of a CodePen editor window displaying a CSS Grid layout. The layout consists of 12 red rectangular elements arranged in a 3x4 grid. The elements are labeled Element 1 through Element 12. Element 2 is a dark blue square element located in the second column of the first row. The other eleven elements are red squares. The CodePen interface shows the HTML, CSS (SCSS), and JS code sections. The CSS section contains the following SCSS code:

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

REPEATED NAMED LINES

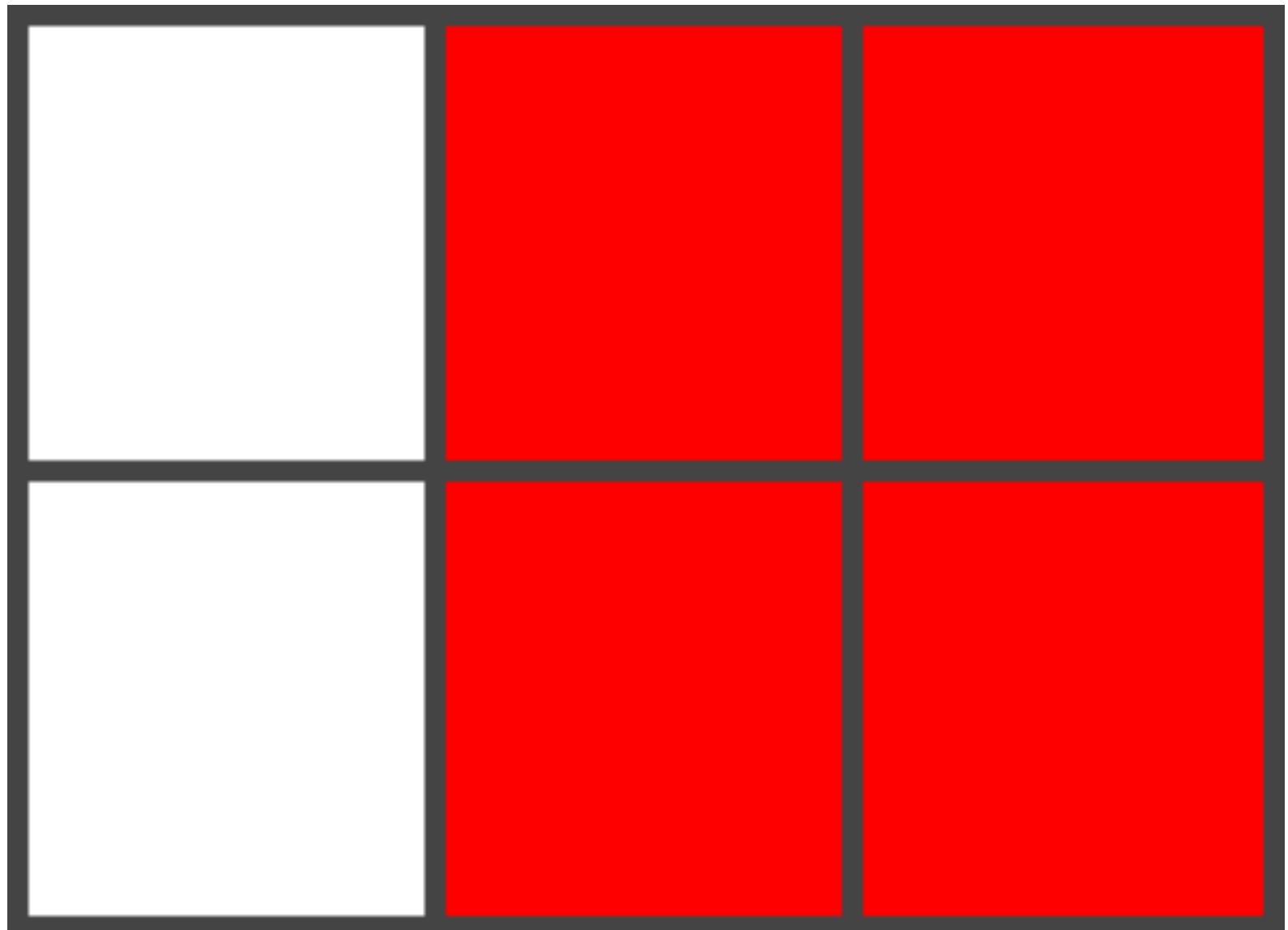
bit.ly/grid_lines3

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;  
}
```

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 CodePen editor interface with a dark theme. The main preview area displays a website layout titled "My Website". The layout consists of three red rectangular sections containing "Article 1", "Article 2", and "Article 3". To the right of these articles is a sidebar with two teal boxes labeled "Ad 1" and "Ad 2". Below the ads is a section titled "Related Posts" containing a list of six items. Further down is a section titled "More Posts" with another list of six items. The footer is a dark bar with a copyright symbol. The right side of the interface features three panels: "HTML" showing the DOM structure, "CSS (SCSS)" showing the SCSS code for styling the grid areas, and "JS" showing the JavaScript code. The CSS panel includes code for the footer, articles, posts, partners, and individual post and partner styles. The JS panel is currently empty.

```
<li>Element 5</li>
<li>Element 6</li>
</ul>
</div>
</aside>

<aside class="partners">
  <div class="item partner">
    <h2>Ad 1</h2>
  </div>
  <div class="item partner">
    <h2>Ad 2</h2>
  </div>
</aside>

<footer class="item footer">
  &copy;
</footer>
```

```
.footer {
  background-color: #17242D;
  grid-area: footer;
}

.articles {
  grid-area: articles;
}

.article {
  margin-bottom: 20px;
}

.posts {
  grid-area: posts;
}

.partners {
  grid-area: partners;
}

.post {
  background-color: #87B3D6;
  margin-bottom: 20px;
}

.partner {
  background-color: #7CCBB0;
  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";  
}
```

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 CodePen editor interface with a dark theme. The main preview area displays a website layout titled "My Website". The layout consists of three red rectangular sections containing "Article 1", "Article 2", and "Article 3". To the right of these articles are two teal rectangular boxes labeled "Ad 1" and "Ad 2". Further down the page are two blue rectangular boxes labeled "Related Posts" and "More Posts", each containing a list of six items. The "Related Posts" box lists "Element 1" through "Element 6", while the "More Posts" box lists "Element 1" through "Element 6". The bottom right corner of the preview area features a small circular icon with a copyright symbol.

The right side of the interface contains three code editors: "HTML", "CSS (SCSS)", and "JS". The "HTML" editor shows the structure of the website with various divs and aside elements. The "CSS (SCSS)" editor contains the following SCSS code:

```
15 }
16
17 .footer {
18   background-color: #17242D;
19   grid-area: footer;
20 }
21
22 .articles {
23   grid-area: articles;
24 }
25
26 .article {
27   margin-bottom: 20px;
28 }
29
30 .posts {
31   grid-area: posts;
32 }
33
34 .partners {
35   grid-area: partners;
36 }
37
38 .post {
39   background-color: #87B3D6;
40   margin-bottom: 20px;
41 }
42
43 .partner {
44   background-color: #7CCBB0;
45   margin-bottom: 20px;
46 }
```

The "JS" editor 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;  
}
```

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 CodePen editor showing a CSS Grid layout example. The layout consists of five red grid items and one blue grid item. The grid has 2 columns and 3 rows. The first column contains a large red item (row 1), a blue item (row 2), and a large red item (row 3). The second column contains two smaller red items (row 1 and row 2). The blue item spans the height of the first two rows. The CSS code uses grid-template-columns and grid-template-rows with magic areas like [content-start] and [aside-end]. The JS tab shows the URL of the pen.

```
HTML
<div class="grid">
  <div class="grid-item item"></div>
  <div class="grid-item item"></div>
  <div class="grid-item item"></div>
  <div class="grid-item item"></div>
  <div class="grid-item item"></div>
</div>
```

```
CSS
.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-gap: 10px;
}

.grid-item:last-child {
  background: #878306;
  grid-area: content;
}
```

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

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;  
}
```

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 CodePen interface with a dark theme. The main preview area displays a grid of nine red rectangular elements labeled Element 1 through Element 9. Element 2 is a larger teal rectangle spanning two columns. The grid is contained within a container with a black border.

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 (SCSS)

```
.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: #7CCBB0;
  grid-area: 2 / 2 / span 2 / span 2;
}
```

JS

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

Console Assets Comments *

Save Fork Settings Change View

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;  
}
```

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";  
}
```

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;  
}
```

A screenshot of a CodePen editor window titled "CSS Grid Layout Lines: grid-area". The page displays a grid layout with 9 red rectangular elements labeled Element 1 through Element 9. Element 2 is a green rectangle spanning two columns. The grid structure is defined by the following CSS:

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

The CSS also includes grid-area definitions for each element:

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

.item:nth-child(2) {
  background-color: #7CCBB0;
  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;  
}
```

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 window shows a 3x2 grid of five items. Item 1 is in the top-left, item 2 is in the top-right, item 3 is in the middle-left, item 4 is in the middle-right, and item 5 is in the bottom-left. Item 5 has a red background. The CodePen interface includes tabs for "HTML", "CSS", and "JS", and a sidebar with "Save", "Fork", "Settings", and "Change View" buttons.

```
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;
}
```

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";  
}
```

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 titled "grid shorthand 2". The window displays a grid layout with five items labeled 1 through 5. Items 1, 2, and 3 are in the top row, while items 4 and 5 are in the bottom row. Item 1 is light red, item 2 is salmon, item 3 is pink, item 4 is salmon, and item 5 is red.

The editor interface includes tabs for "HTML", "CSS", and "JS". The "HTML" tab shows the following structure:

```
<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>
```

The "CSS" tab contains the following CSS code:

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

.grid {
  [first] "header header header" 200px
  [second] "..... footer footer" 100px [last] /
  [first] 1fr [second] 1fr [third] 1fr [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;
}
```

The "JS" tab is currently empty.

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 red box containing a heading and some lorem ipsum text. The HTML panel shows a single div with a grid item. The CSS panel shows a grid class with a 400px / 500px ratio and a grid item class with a salmon background color and 20px padding. The JS panel is empty.

```
HTML
1 <div class="grid">
2   <div class="grid_item">
3     <h2>Heading</h2>
4     <p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Omnis sunt expedita labore dicta quia nostrum magnam perferendis quam accusamus cupiditate unde ullam facilis quisquam, soluta accusantium a sit nihil?</p>
5   </div>
6 </div>

CSS
1 .grid {
2   display: grid;
3   grid: 400px / 500px;
4 }
5
6 .grid_item {
7   background-color: salmon;
8   padding: 20px;
9   /*align-self: end;*/
10 }
```

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

CSS Intrinsic And Extrinsic Sizing

- The module defines additional ways to control size.

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.

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.**

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 right side of the window displays the code:

```
1 <h1>min-content</h1>
2
3 <blockquote>
4   Use <code>width: min-content</code> 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.
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
8 <div class="wrapper">
9   <div class="min">Lorem ipsum dolor sit amet consectetur
   adipisicing elit.
10  </div>
11 </div>
```

```
border-color: #17242B;
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;
```

At the bottom, there are tabs for "Console", "Assets", "Comments", and "JS". The status bar says "Last saved less than a minute ago".

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 page content is titled "max-content" and contains a blockquote with a red border and white background, containing the text "Lorem ipsum dolor sit amet consectetur adipisicing elit.". The sidebar on the right displays the HTML and CSS code for this snippet.

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
8 <div class="max-content">
9   ...
10</div>
```

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;
```

JS

```
</script>
```

At the bottom, there are tabs for "Console", "Assets", "Comments", and "JS". The status bar indicates "Last saved less than a minute ago".

MAX-CONTENT

bit.ly/grid_max

A screenshot of a CodePen editor window titled "min-content grid layout". The URL is <https://codepen.io/matuzo/pen/GQKZqN?editors=1100>. The page displays a grid layout with six items: Alfa, Bravo (with additional content), Charlie, Delta, Echo, and Foxtrot. The Bravo item wraps onto a new line due to its content. The CSS defines a grid with three columns and a border of 2px solid #17242D, a width of 500px, and a grid-gap of 20px. The grid-template-columns are set to min-content, min-content, and min-content.

```
HTML
1 <div class="grid">
2   <div class="item">Alfa</div>
3   <div class="item">Bravo has some additional content which will wrap</div>
4   <div class="item">Charlie</div>
5   <div class="item">Delta</div>
6   <div class="item">Echo</div>
7   <div class="item">Foxtrot</div>
8 </div>
```

```
CSS
1 .grid {
2   border: 2px solid #17242D;
3   width: 500px;
4   display: grid;
5   grid-gap: 20px;
6   grid-template-columns: min-content min-content min-content;
7 }
```

```
JS
```

MIN-CONTENT FOR TRACK SIZING

bit.ly/grid_min2

A screenshot of a CodePen editor window titled "max-content grid layout". The URL is <https://codepen.io/matuzo/pen/zROqoX?editors=1100>. The title bar shows the project name and a save button. The sidebar includes options for "Save", "Fork", "Settings", and "Change View". The main content area displays a grid layout with six items: "Alfa", "Bravo has some additional content which will wrap", "Charlie", "Delta", "Echo", and "Foxtrot". The "Bravo" item contains a multi-line text block. The "HTML" panel shows 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 shows the following SCSS code:

```
.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;
}
```

The "JS" panel is empty. At the bottom, there are links for "Console", "Assets", "Comments", and "Last saved less than a minute ago".

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 page URL is <https://codepen.io/matuzo/pen/MQgywB?editors=1100>. The editor interface includes a header with "Save", "Fork", "Settings", and "Change View" buttons, and a sidebar with "Console", "Assets", "Comments", and a search bar.

The preview area shows a grid layout with five red rounded rectangular cards:

- Alfa
- Bravo has some additional content which will wrap
- Charlie
- Delta
- Echo
- Foxtrot

The HTML code defines a grid with six items:

```
<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) code styles the grid container:

```
.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 section is empty.

FIT-CONTENT

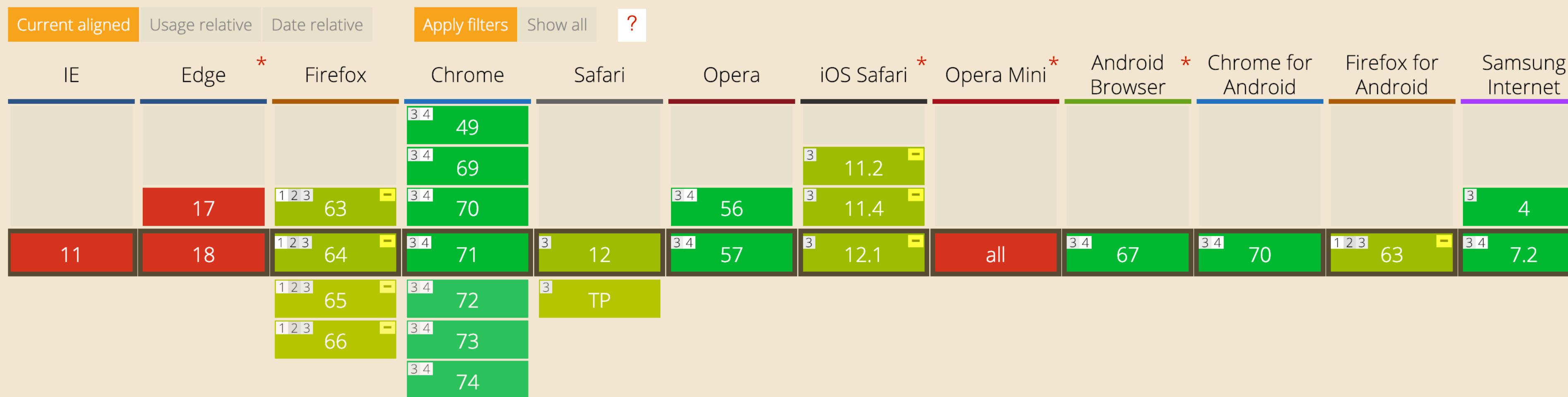
bit.ly/grid_fit

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	% of all users
Global	69.83% + 18.27% = 88.11%
unprefixed:	68.04% + 2.16% = 70.19%



PATTERNS

Some common patterns



EXERCISE 4

The screenshot shows a web browser window with a dark blue header bar containing the word "Header". Below the header is a white content area with five identical paragraphs of placeholder text (Lorem ipsum) stacked vertically. At the bottom of the page is a dark blue footer bar with the copyright notice "© 2017". The browser interface includes a search bar, a tab labeled "CodePen - CSS Grid Layout Stic", and various control icons.

Header

© 2017

Search or enter address

CodePen - CSS Grid Layout Stic

III IV ⓘ T ABP ⌂ ⌂

© 2017

© 2017

© 2017

© 2017

© 2017

© 2017

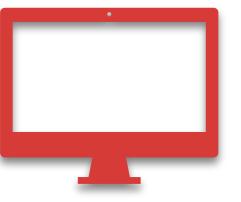


EXERCISE 4

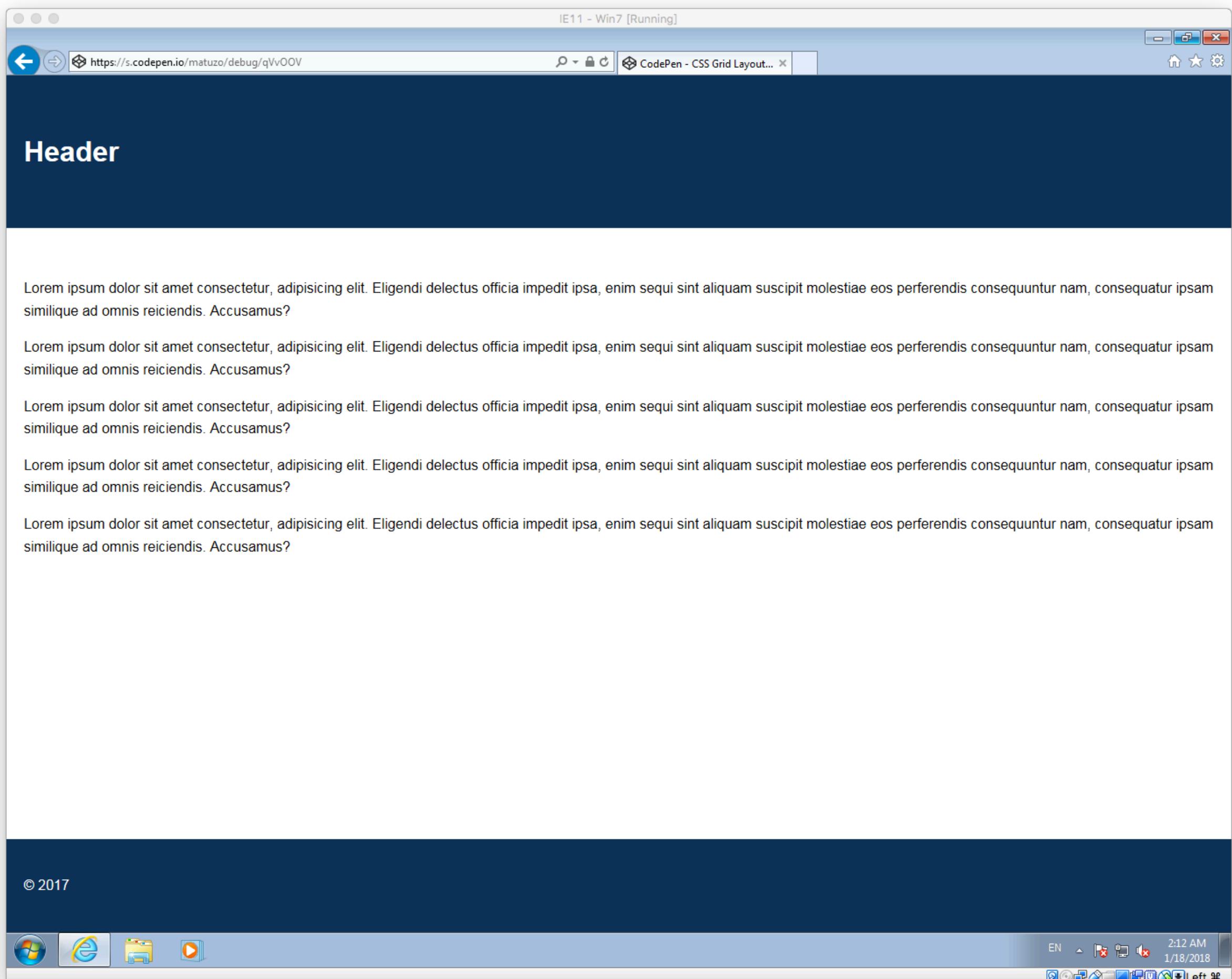
Sticky footer

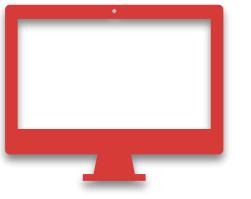
- Try to implement a sticky footer pattern with grid.
- Footer Sticky at the bottom of the page.
- Footer moves with the content if the content is longer than the page.
- Flexible height for header and footer.

https://bit.ly/grid_sticky

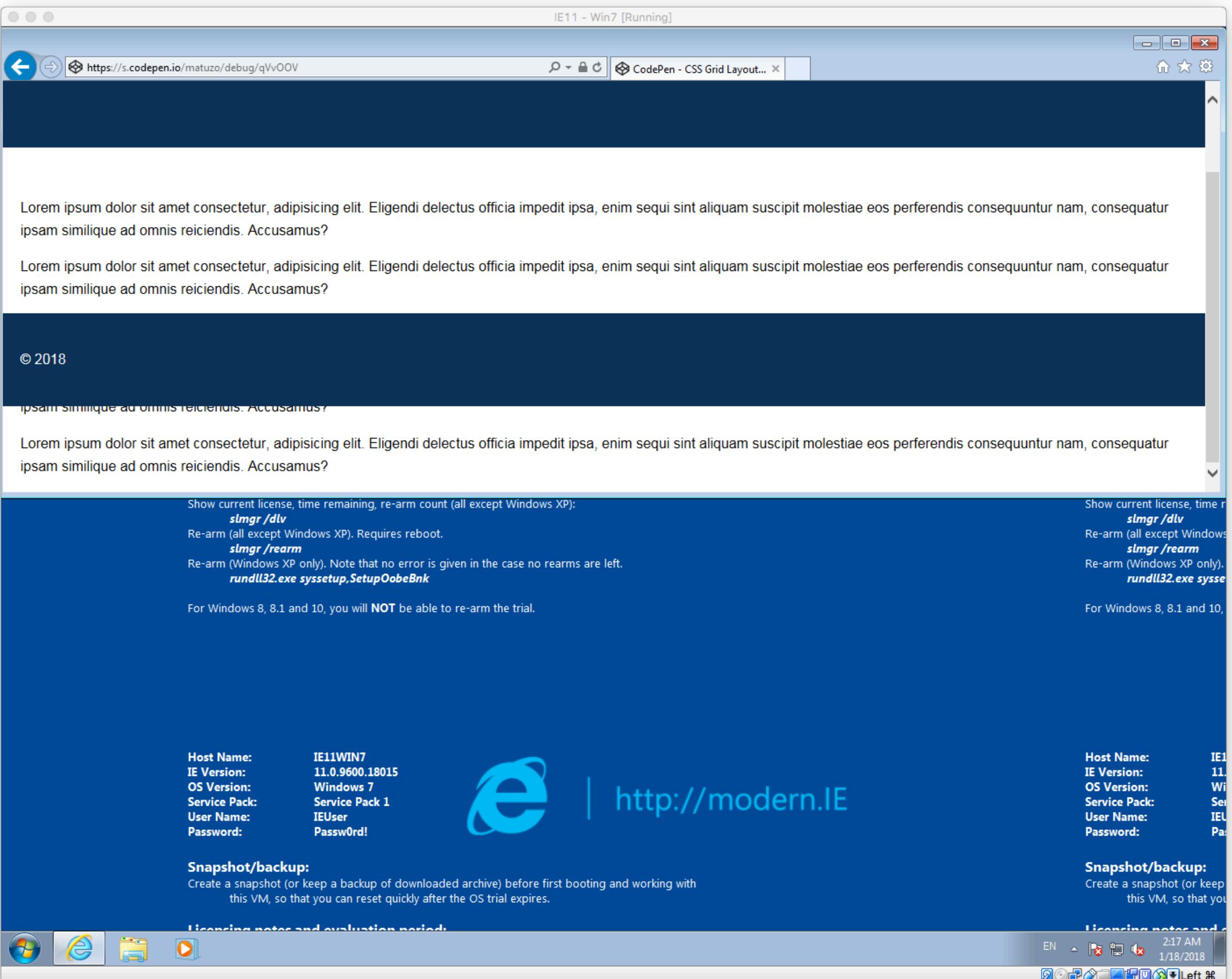


EXERCISE 4.5





EXERCISE 4.5



STICKY FOOTER IN IE: 😞

https://bit.ly/grid_sticky_ie



EXERCISE 5

The screenshot shows a web page titled "My blog". The layout consists of a series of repeating horizontal sections. Each section has a dark blue background. Within each section, there is a red rectangular box containing a heading and some placeholder text ("Lorem ipsum..."). The sections are separated by white space.

- Section 1:** Red box contains "Heading" and "Lorem ipsum dolor sit amet consectetur adipisicing elit."
- Section 2:** Dark blue background contains text: "Lorem ipsum dolor sit amet consectetur adipisicing elit. Sint expedita ea quae eaque perferendis? Dolor recusandae voluptatibus exercitationem cumque, quod voluptatem maiores ab iure reprehenderit ullam mollitia temporibus quos dolores."
- Section 3:** Red box contains "Heading" and "Lorem ipsum dolor sit amet consectetur adipisicing elit."
- Section 4:** Red box contains "Heading" and "Lorem ipsum dolor sit amet consectetur adipisicing elit."
- Section 5:** Dark blue background contains text: "Lorem ipsum dolor sit amet consectetur adipisicing elit. Sint expedita ea quae eaque perferendis? Dolor recusandae voluptatibus exercitationem cumque, quod voluptatem maiores ab iure reprehenderit ullam mollitia temporibus quos dolores."
- Section 6:** Red box contains "Heading" and "Lorem ipsum dolor sit amet consectetur adipisicing elit."
- Section 7:** Red box contains "Heading" and "Lorem ipsum dolor sit amet consectetur adipisicing elit."
- Footer:** A dark blue footer bar at the bottom of the page.

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 {
```

```
}
```

A grid with as many 180px wide columns as possible

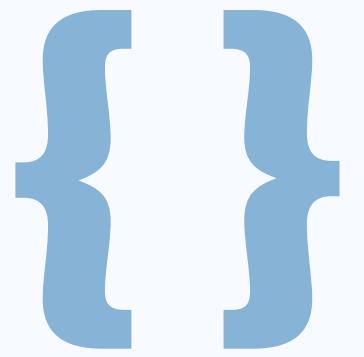
{ }

```
ul {  
  display: grid;  
}  
A grid with as many 180px wide columns as possible
```

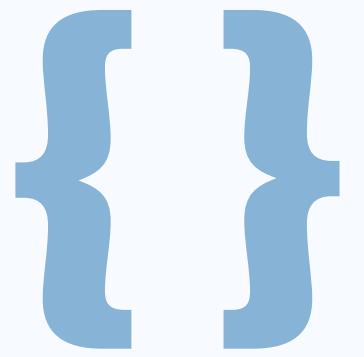
{ }

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

A grid with as many 180px wide columns as possible

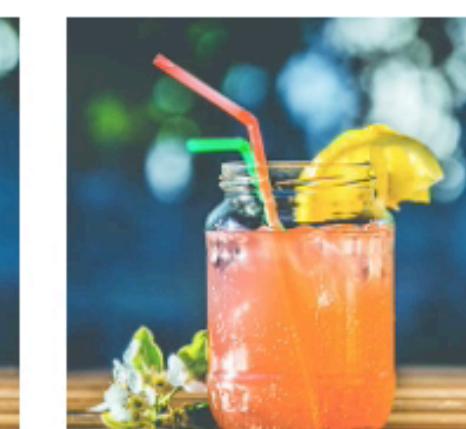


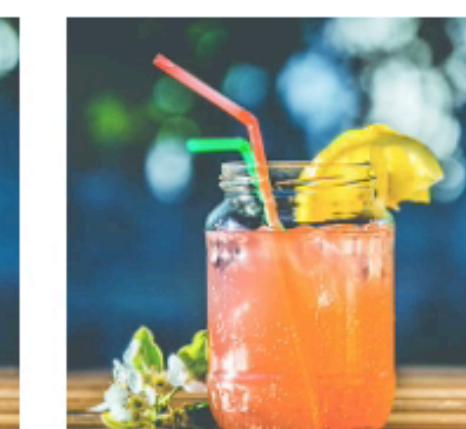
```
ul {  
  display: grid;  
  grid-template-columns: repeat(auto-fill, minmax(180px, 1fr));  
  grid-gap: 20px;  
}
```



```
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;  
}
```

{ }

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

{ }

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

{ }

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

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

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

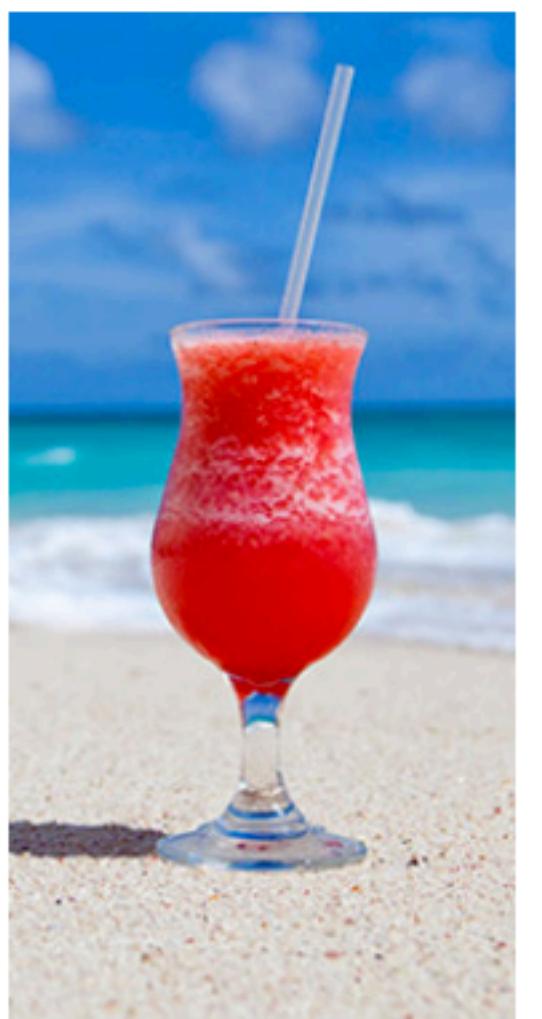
```
.long {  
  grid-row: span 2;  
}
```

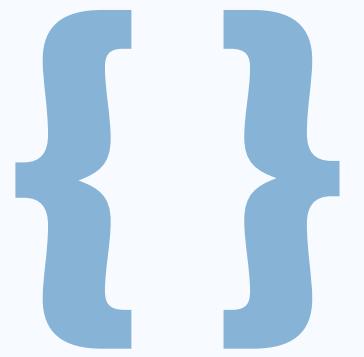
{ }

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

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

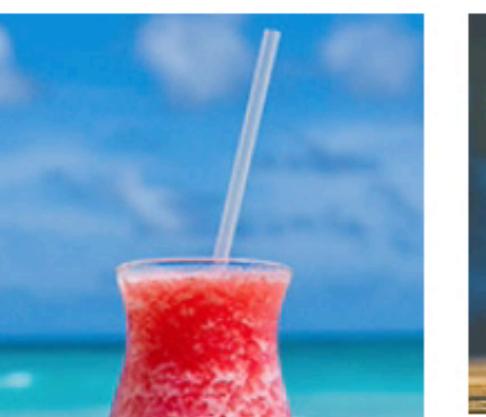
```
.large2 {  
  grid-column: 1 / span 2;  
}
```

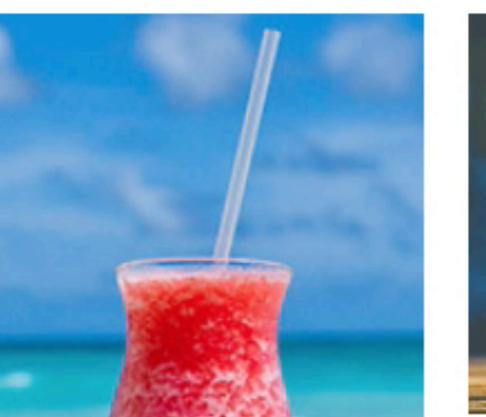
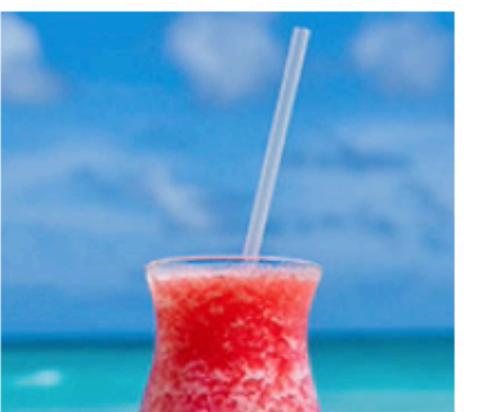




```
ul {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(180px, 1fr));
  grid-gap: 20px;
  grid-auto-rows: 180px;
  grid-auto-flow: dense;
}
```

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









Accessibility

Accessibility

- Planing and design start with the content and source order.

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.

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

Development process

- Start with a semantic and well structured document.

Development process

- Start with a semantic and well structured document.
- Create the grid and check if the structure still makes sense.

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.

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

Stacking order

- The rendering order of elements on the z-axis can be influenced by changing the `z-index` property.

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*.

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

Stacking context

- position with a value of relative, absolute, fixed or sticky.

Stacking context

- position with a value of relative, absolute, fixed or sticky.
- opacity less than 1.

Stacking context

- position **with a value of relative, absolute, fixed or sticky.**
- opacity **less than 1.**
- mix-blend-mode **value other than normal.**

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

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

Stacking context

- Elements with a `-webkit-overflow-scrolling` value touch.

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.

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.

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 CodePen interface with the title "CSS Stacking order". The main content area displays a vertical stack of 15 colored boxes (red, blue, green, yellow, black) labeled 1 through 15. A red button labeled "ON/OFF" is positioned above the stack. The CSS code uses nth-child and nth-of-type pseudo-selectors to apply various effects to specific boxes:

```
button.ON/OFF
```

```
HTML
<button>ON/OFF</button>
<div>
  <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>
```

```
CSS (SCSS)
.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;
  }
}
```

At the bottom of the screen, there are tabs for "Console", "Assets", "Comments", and a search bar. On the right side, there are buttons for "Save", "Fork", "Settings", "Change View", and user 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
```

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

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 interface with the title "Basic grid system functionality with CSS Grid Layout". The preview area displays a 12-column grid of red boxes labeled 1 through 12. The sidebar contains the following code:

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

At the bottom, there are tabs for Console, Assets, Comments, and a file icon. On the right, there are buttons for Save, Fork, Settings, Change View, and a user profile. A message at the bottom right says "Last saved 8 months ago".

BASIC GRID SYSTEM FUNCTIONALITY

bit.ly/grid_mq3

The screenshot shows a CodePen editor window with the title "CSS Grid Layout Responsive Webdesign". The preview area displays a grid layout with six items labeled 1 through 6. Items 1, 2, and 3 are in the first row, while 1, 2, 3, 4, and 5 are in the second row, with item 6 being a single column item below them. The CodePen interface includes tabs for "HTML", "CSS (SCSS)", and "JS", along with various toolbars and status messages.

```
HTML
1 <div class="grid">
2   <div class="item">
3     <h2>1</h2>
4   </div>
5   <div class="item">
6     <h2>2</h2>
7   </div>
8   <div class="item">
9     <h2>3</h2>
10  </div>
11 </div>
12
13 <div class="grid">
14   <div class="item">
15     <h2>1</h2>
16   </div>
17   <div class="item">
18     <h2>2</h2>
19   </div>
20   <div class="item">
21     <h2>3</h2>
22   </div>
23   <div class="item">
24     <h2>4</h2>
25   </div>
26   <div class="item">
27     <h2>5</h2>
28   </div>
29   <div class="item">
30     <h2>6</h2>
31   </div>
32 </div>

CSS (SCSS)
1 $site_max_width: 1200px;
2 $gap: 20px;
3 $max_columns: 4;
4
5 // $min-width: ($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 }

JS
```

RWD WITHOUT MEDIA QUERIES

bit.ly/grid_mq4

PROGRESSIVE ENHANCEMENT

Progressive Enhancement

Progressive Enhancement

- Progressive enhancement is a philosophy.

Progressive Enhancement

- Progressive enhancement is a philosophy.
- One principle: fault tolerance.

Progressive Enhancement

- Progressive enhancement is a philosophy.
- One principle: fault tolerance.
- Browsers must ignore anything they don't understand in HTML and CSS.

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.

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>



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



<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

Progressive Enhancement

- PE doesn't require that you provide the same experience in different browsers.

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.

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

- 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.

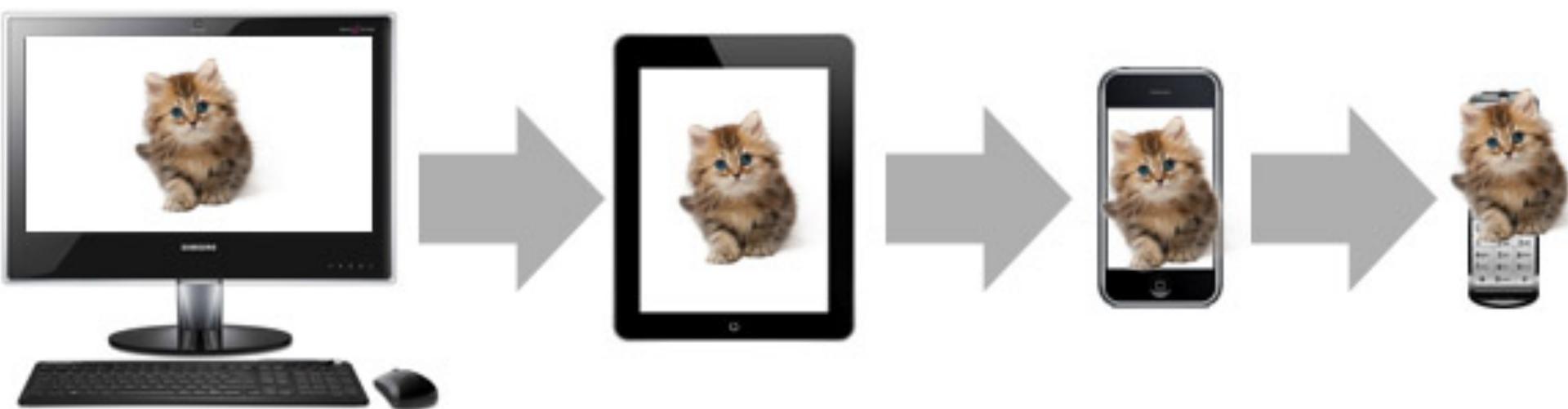
```
div {  
    width: 50vw;  
    width: 50vmax;  
}
```

Progressive Enhancement in CSS

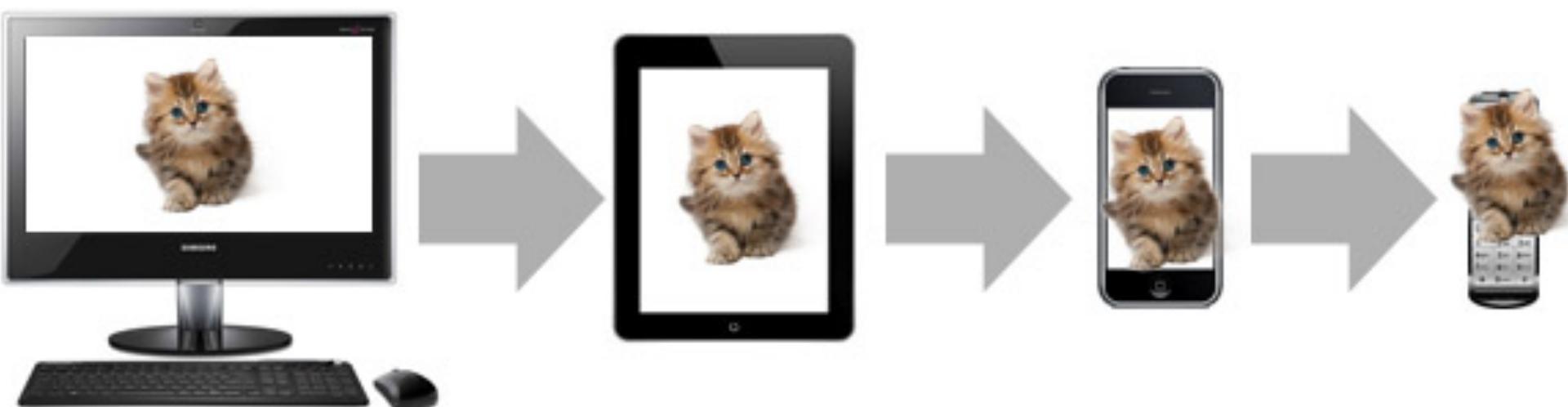
By preferring (min-width) over max-width in media queries, not only smartphone users but users of browsers than can't interpret media queries will get at least some kind of experiences.

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

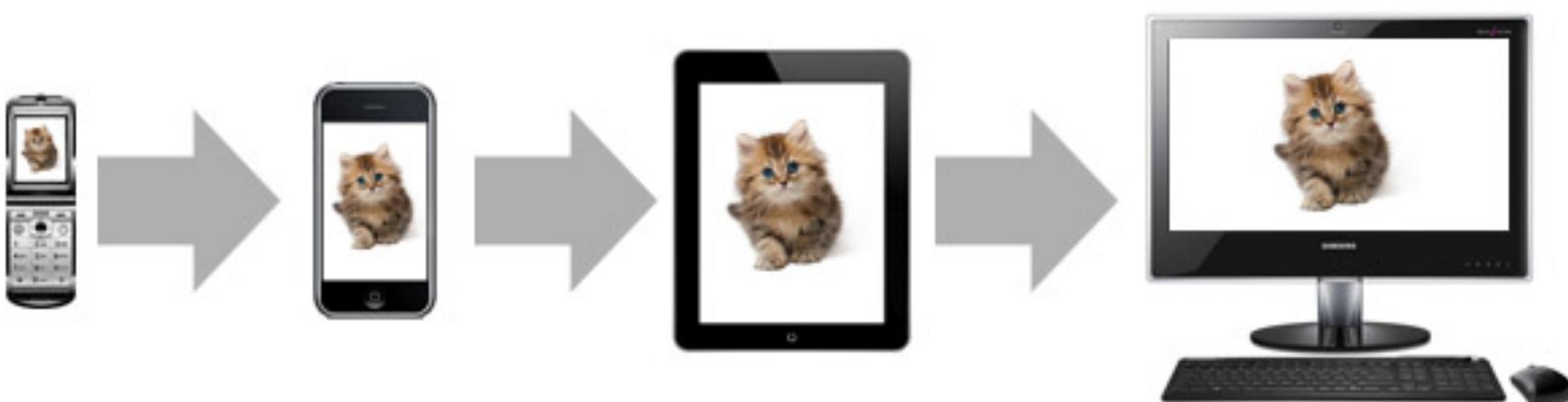
Graceful Degradation



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

Progressive Enhancement

- **float** and **clear** have no effect

Progressive Enhancement

- **float** and **clear** have no effect
- **vertical-align** has no effect

Progressive Enhancement

- `float` and `clear` have no effect
- `vertical-align` has no effect
- `flex` items become grid items

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.

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);  
}
```

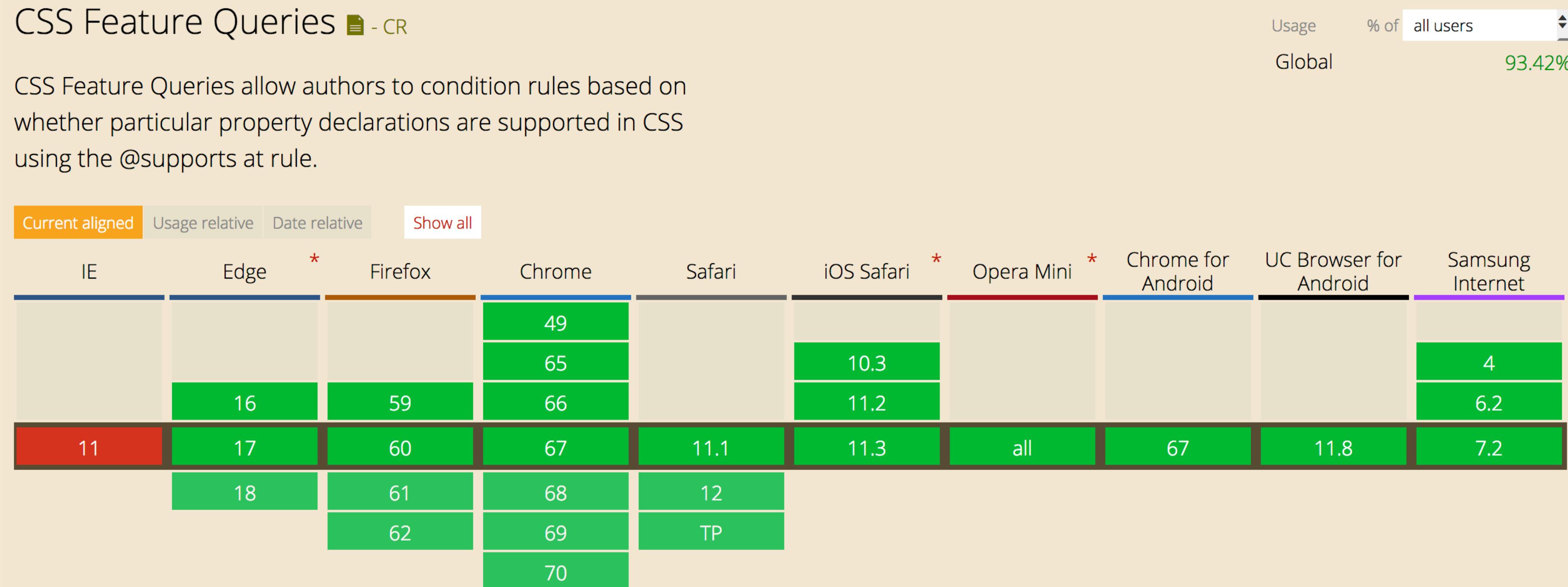
Progressive Enhancement

Feature Queries.

Check for property + value.

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

Feature Queries



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

Detecting Support

Detecting Support

- We don't need tools like Modernizr.

Detecting Support

- We don't need tools like Modernizr.
- All browsers that understand Grid understand Feature Queries as well.

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.

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;  
  }  
}
```

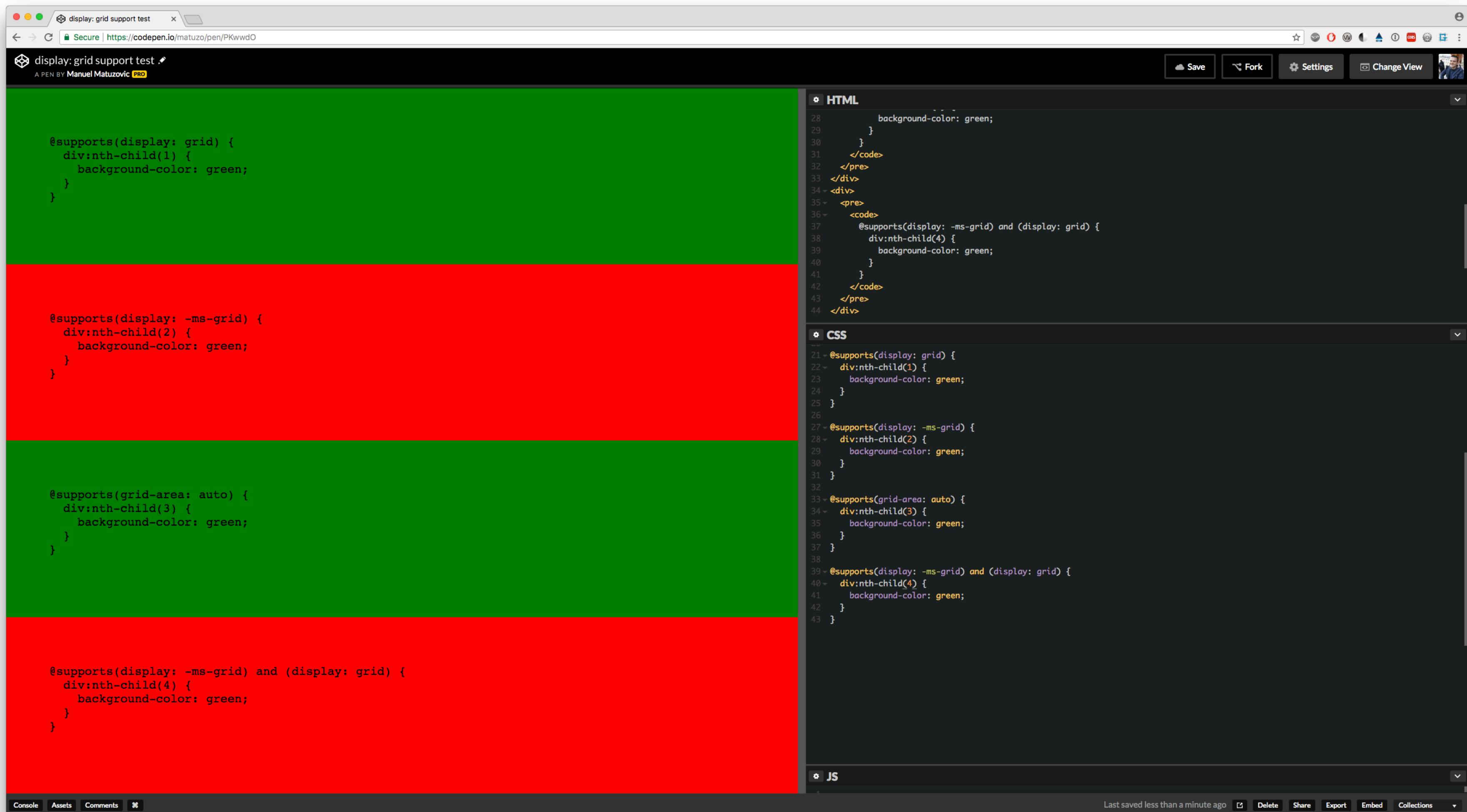
Feature detection

Detecting the new spec

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

Detecting the old spec

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



GRID FEATURE TEST IN CHROME 66

bit.ly/grid_feature1

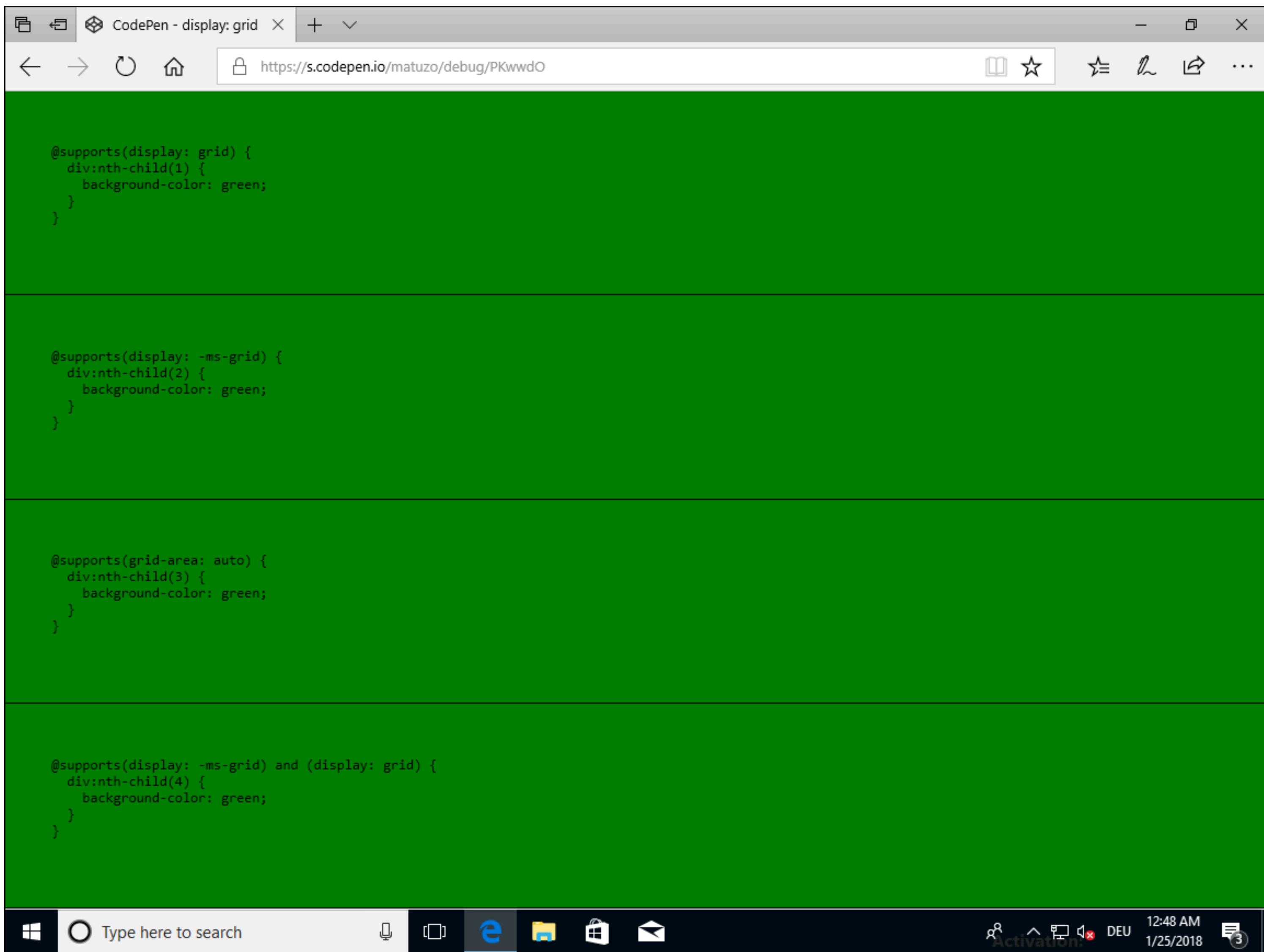
```
@supports(display: grid) {  
  div:nth-child(1) {  
    background-color: green;  
  }  
}
```

```
@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;  
    }  
}
```

CodePen - Demo: Progressive E X +

https://s.codepen.io/matuzo/debug/Emddvx

Enhance!

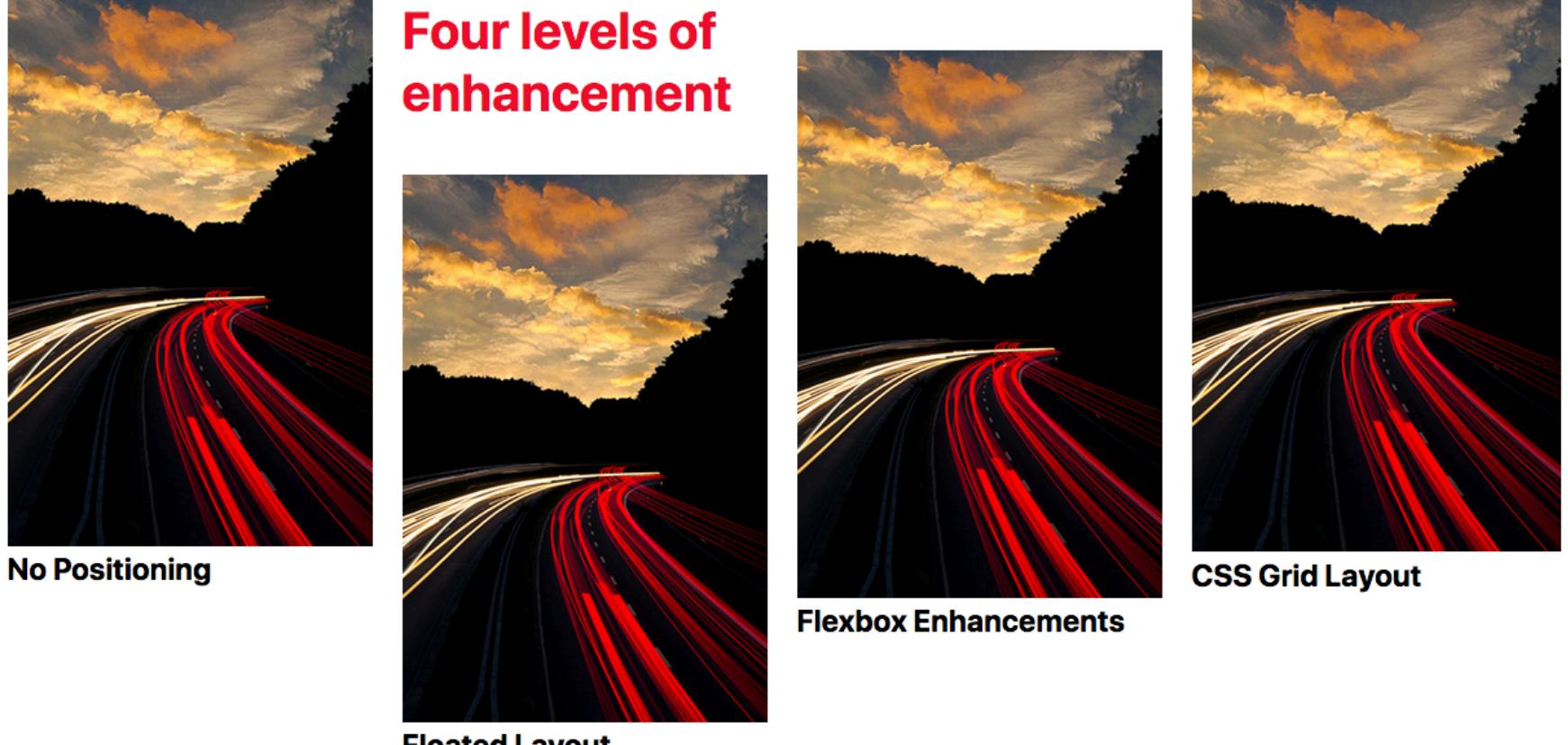
Use Grid today!

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

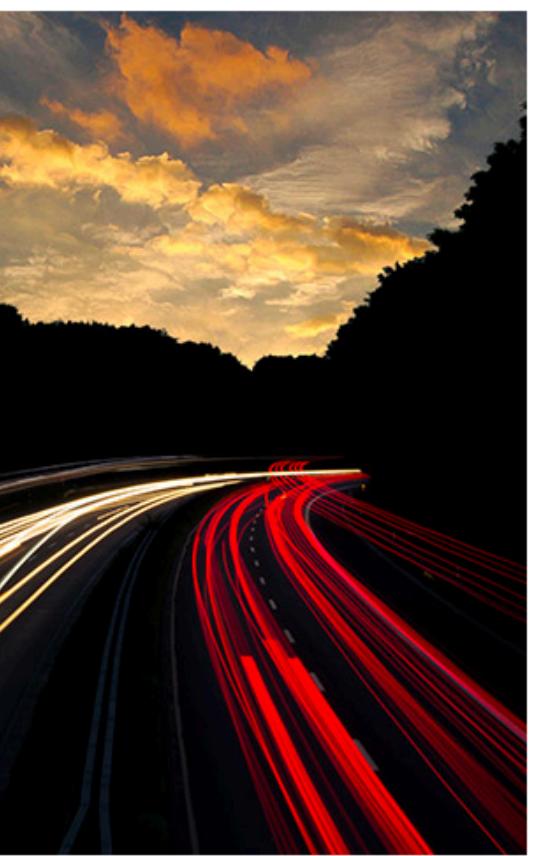
Markup: section, heading
and 4 articles

```
<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



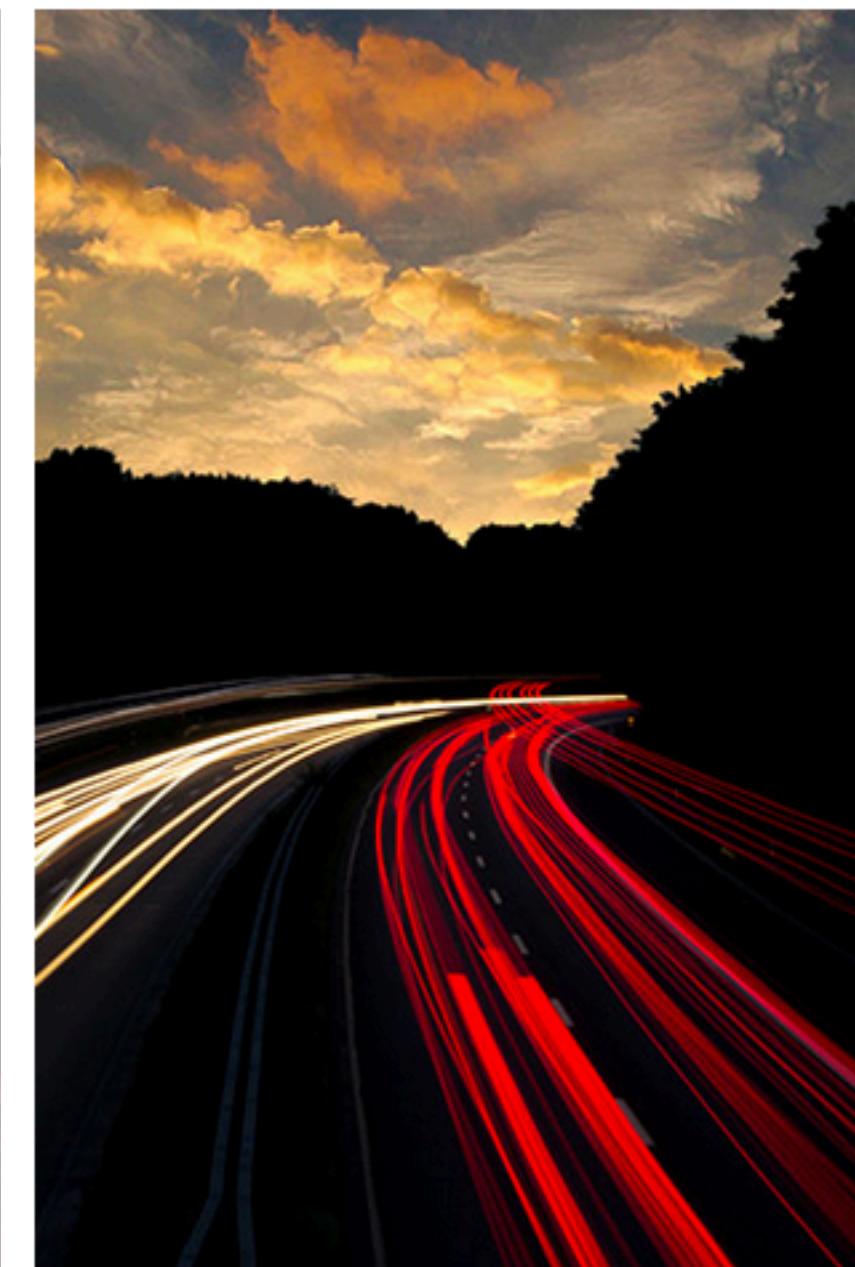
Floated Layout



Flexbox Enhancements



CSS Grid Layout



Progressive Enhancement Example

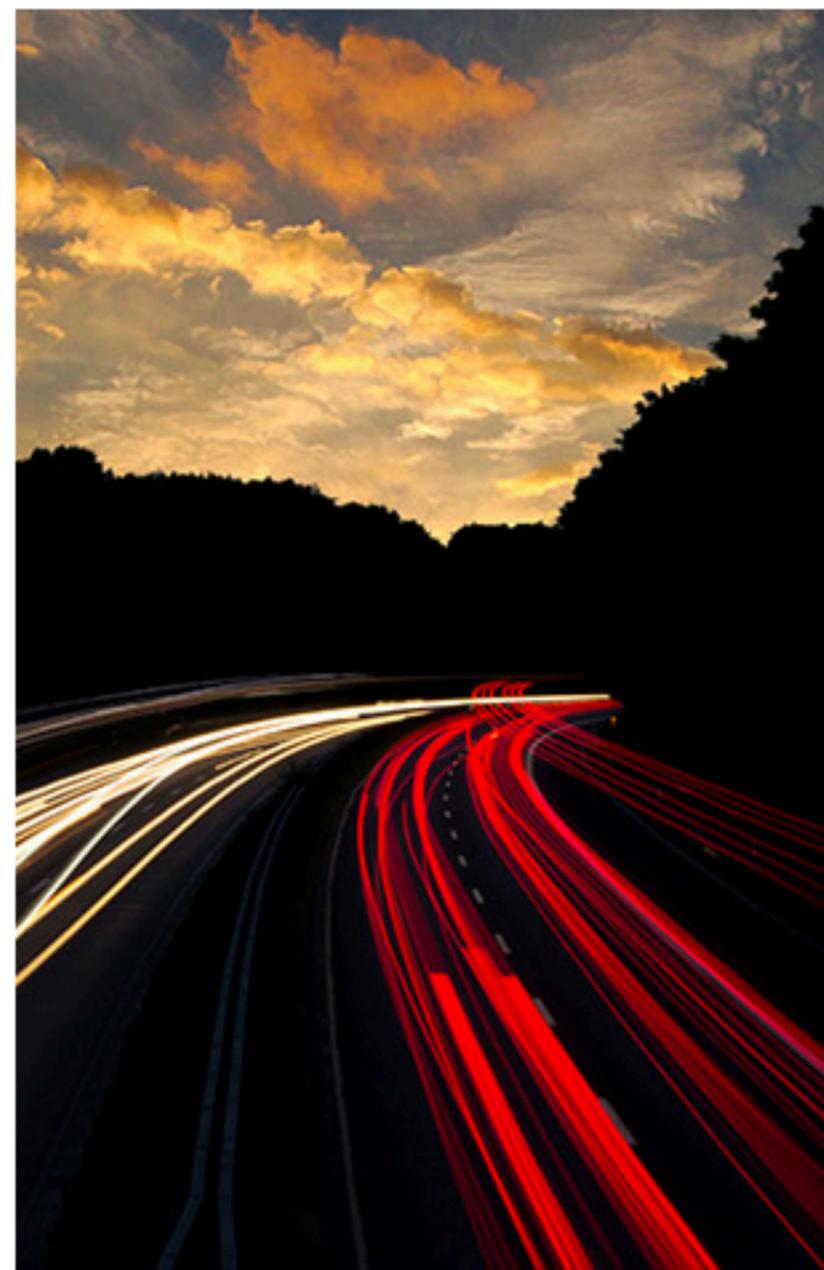
Enhancing with Flexbox.

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

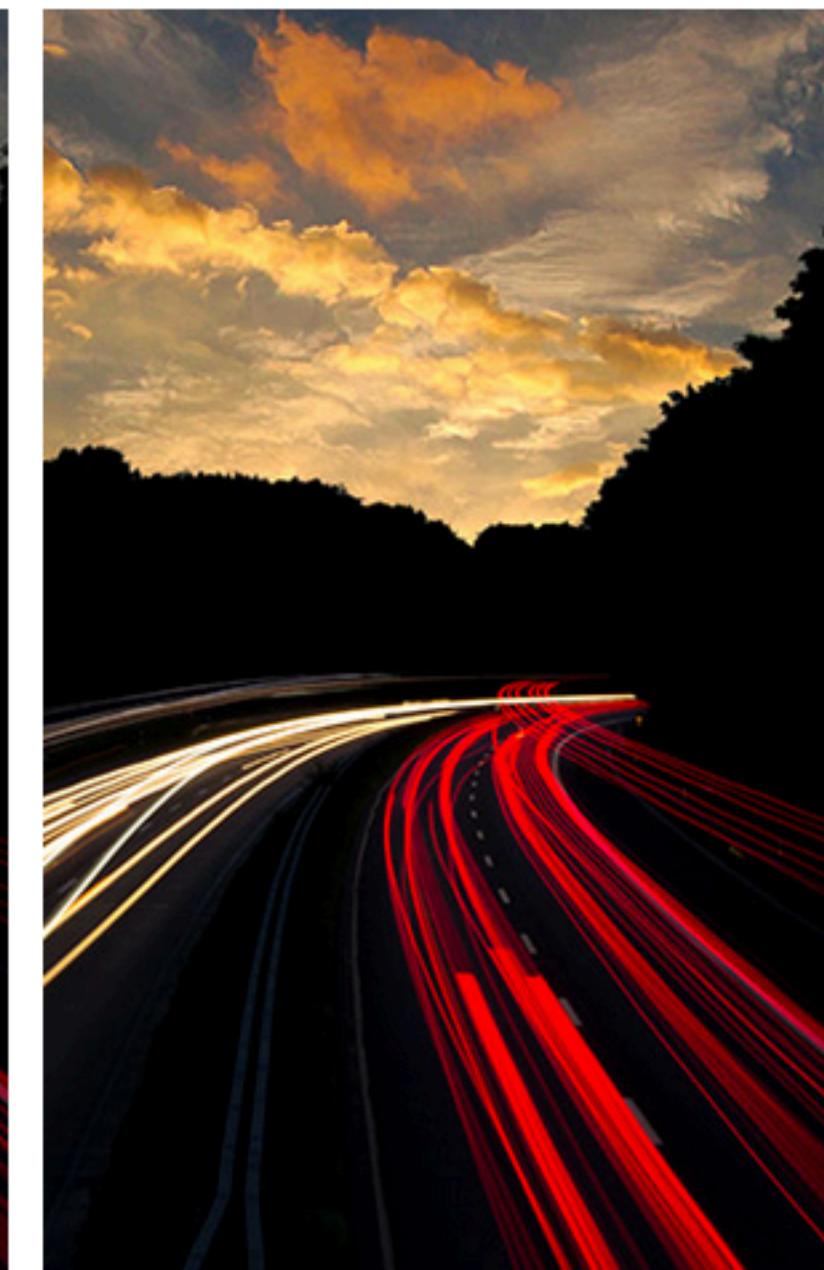
Four levels of enhancement



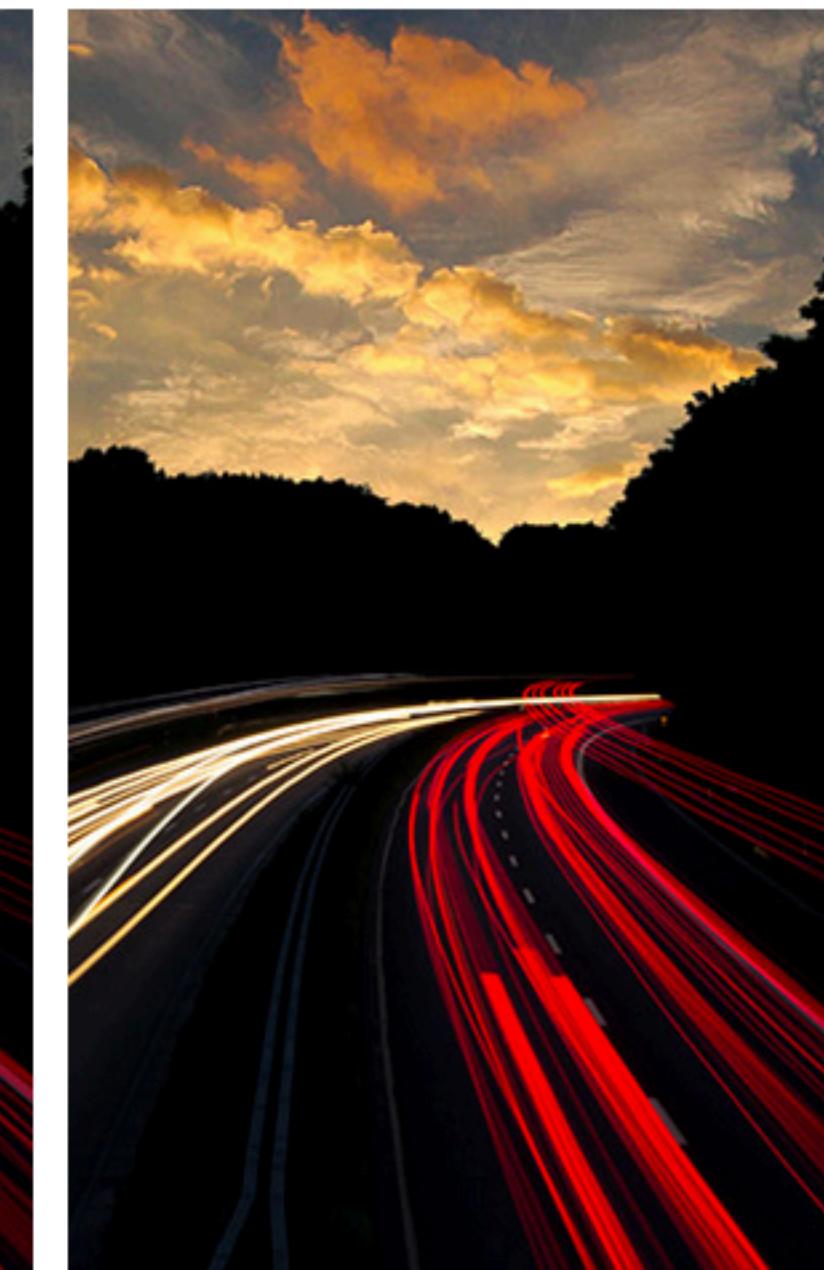
No Positioning



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

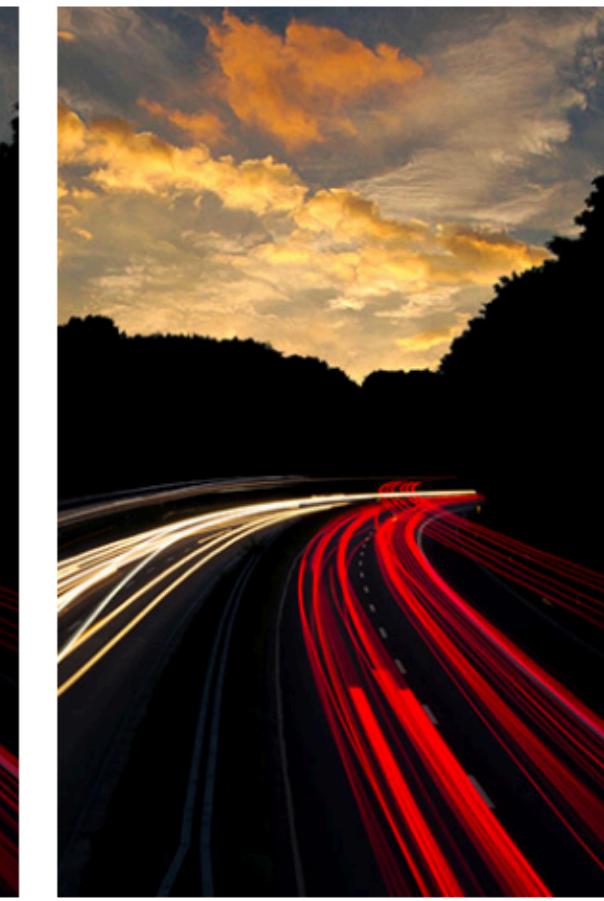
Overwrite only the properties that actually need overwriting.

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

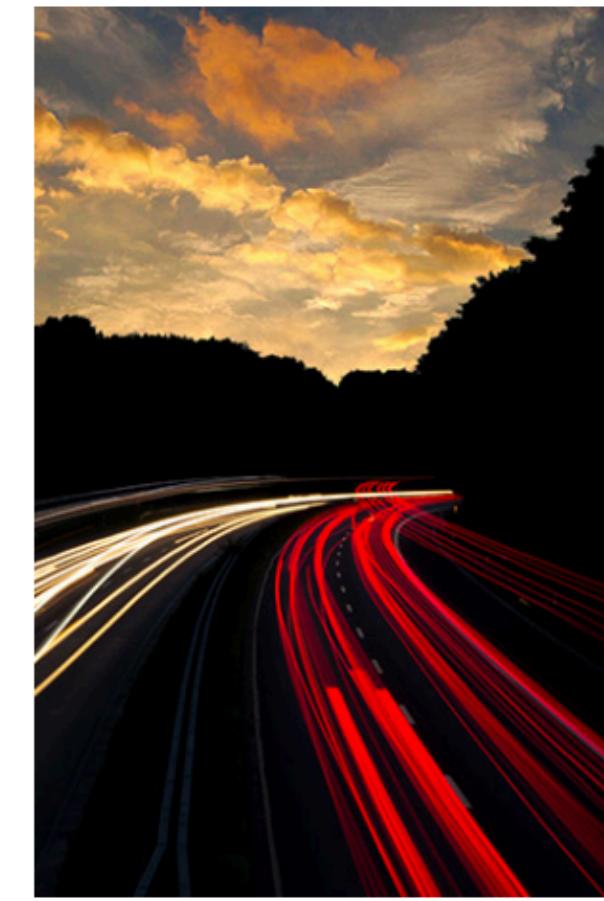
Four levels of enhancement



No Positioning



Floated Layout



Flexbox Enhancements

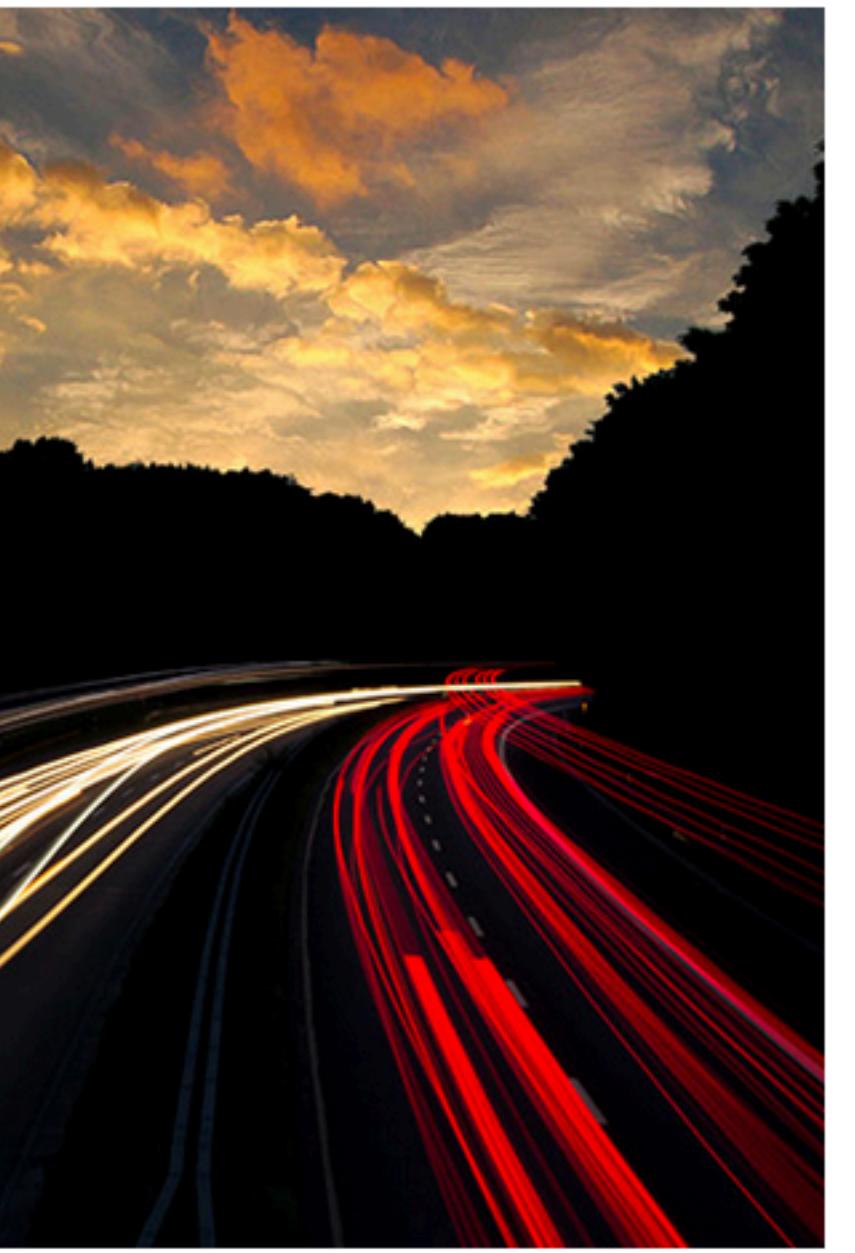


CSS Grid Layout

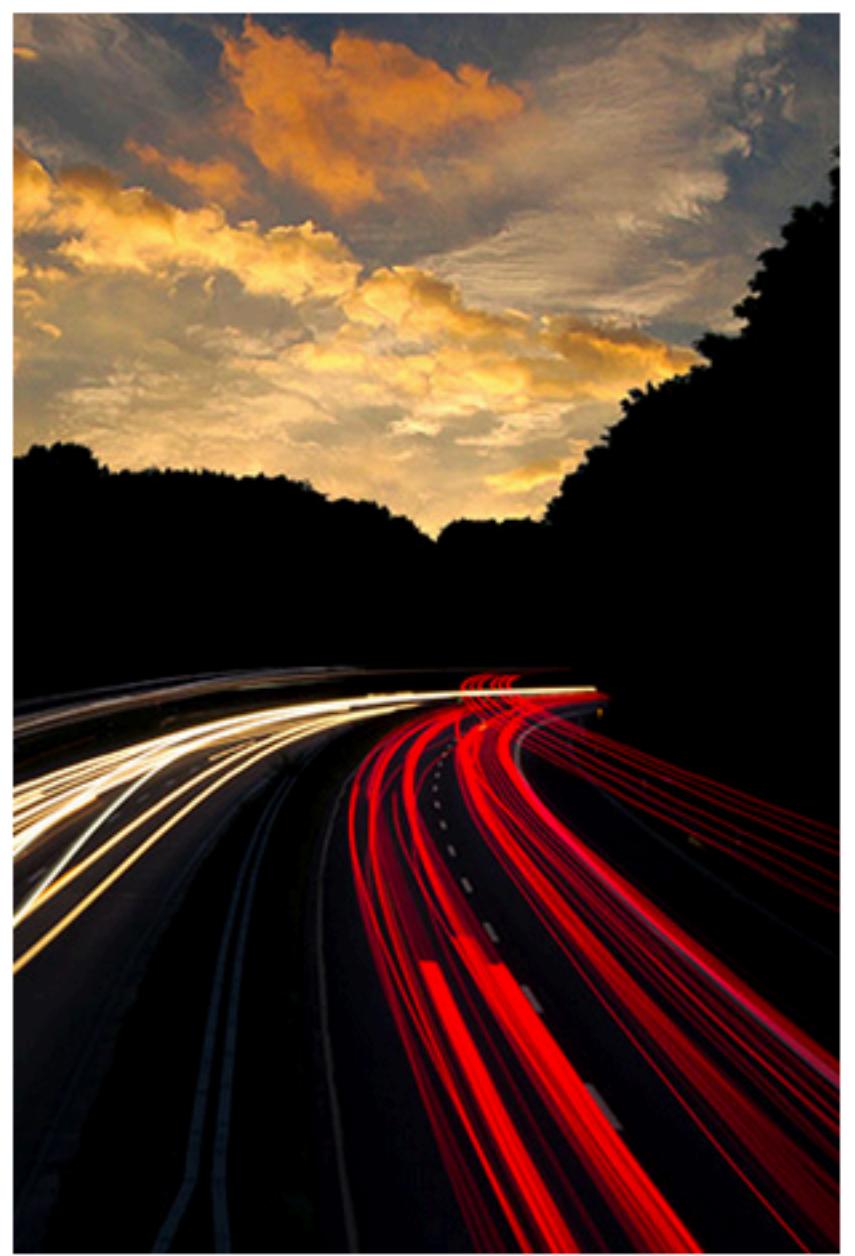
Progressive Enhancement Example

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

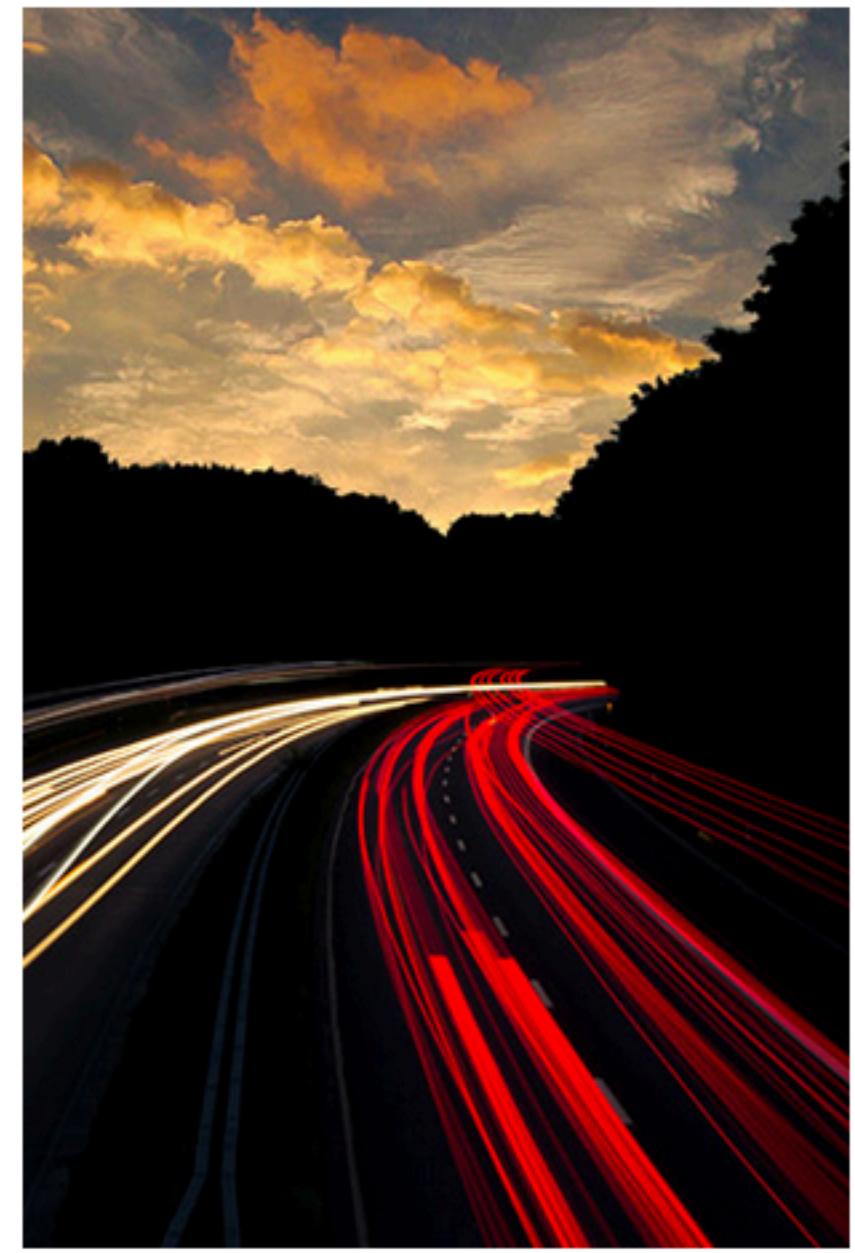


Floated Layout

Four levels of enhancement



Flexbox Enhancements



CSS Grid Layout

CodePen - Demo: Progressive Enhancement

https://s.codepen.io/matuzo/debug/Emddvx

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.

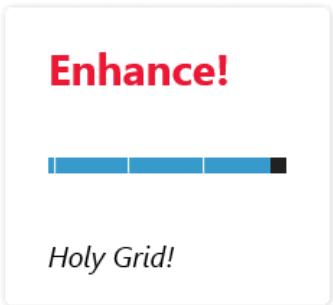
Four levels of enhancement

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

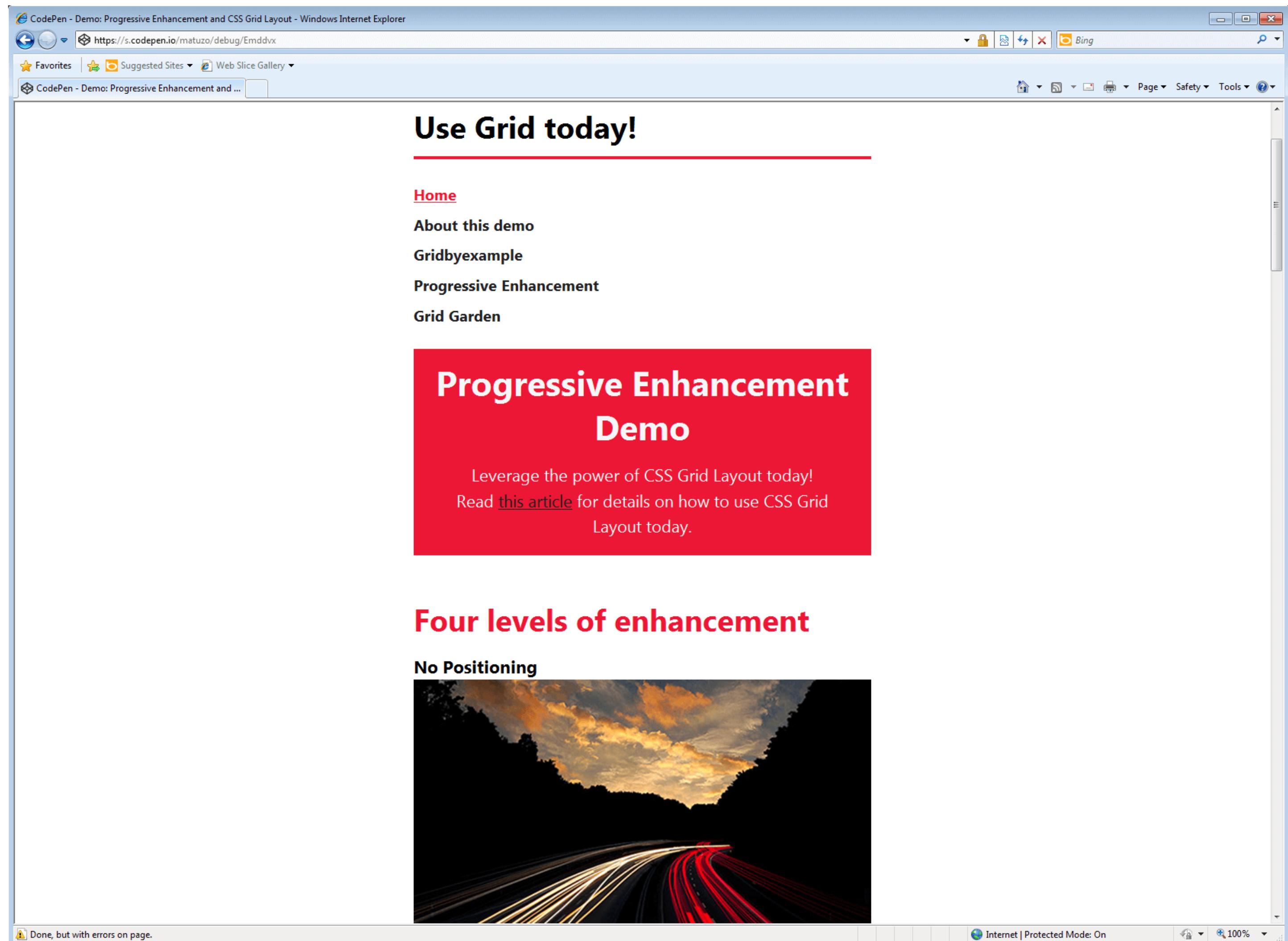


Flexbox Enhancements



CSS Grid Layout





Progressive Enhancement

Progressive Enhancement

- Use vendor prefixes only sparingly, if ever.

Progressive Enhancement

- Use vendor prefixes only sparingly, if ever.
- Grid is build in a way to coexist with other properties.

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.

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.

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

Grid properties in IE10 - Edge 15

Grid properties in IE10 - Edge 15

Level 1 Property	IE 10 Implementation

Grid properties in IE10 - Edge 15

Level 1 Property	IE 10 Implementation
grid-template-columns	-ms-grid-columns

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 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 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 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 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 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

Grid properties in IE10 - Edge 15

Grid properties in IE10 - Edge 15

Level 1 Property	IE 10 Implementation

Grid properties in IE10 - Edge 15

Level 1 Property	IE 10 Implementation
grid	X

Grid properties in IE10 - Edge 15

Level 1 Property	IE 10 Implementation
grid	X
grid-row-start	-ms-grid-row

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 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 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 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 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

Grid properties in IE10 - Edge 15

Grid properties in IE10 - Edge 15

Level 1 Property	IE 10 Implementation

Grid properties in IE10 - Edge 15

Level 1 Property	IE 10 Implementation
grid-area	X

Grid properties in IE10 - Edge 15

Level 1 Property	IE 10 Implementation
grid-area	X
grid-row-gap	X

Grid properties in IE10 - Edge 15

Level 1 Property	IE 10 Implementation
grid-area	X
grid-row-gap	X
grid-column-gap	X

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

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

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?

Is prefixing grid properties worth it?

- No auto-placement

Is prefixing grid properties worth it?

- No auto-placement
- No grid-gap

Is prefixing grid properties worth it?

- No auto-placement
- No grid-gap
- No areas

Is prefixing grid properties worth it?

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

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

Automatic Minimum Size of Grid Items

A PEN BY **Manuel Matuzovic** PRO

Save Fork Settings Change View

Heading

Heading

Heading

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

Console Assets Comments *

AUTOMATIC MINIMUM SIZE OF GRID ITEMS

bit.ly/grid_sizing

Automatic Minimum Size of Grid Items

Automatic Minimum Size of Grid Items

- By default, a grid item cannot be smaller than the size of its content.

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`.

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.

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

Nesting

- A grid item can also be a grid container.

Nesting

- A grid item can also be a grid container.
- Parent grid properties don't apply for nested grid items.

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.

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

Animation

- There are 5 animatable grid properties:

Animation

- There are 5 animatable grid properties:
 - `grid-gap`, `grid-row-gap`, `grid-column-gap` as length, percentage, or calc.

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.

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	✓	✗	✗

The screenshot shows a CodePen project titled "CSS Grid Layout: Animation" by Manuel Matuzovic. The interface includes a header with navigation, save, fork, settings, and change view buttons. The main content area displays a grid of 12 red cards labeled "Element 1" through "Element 12". A blue "ANIMATE" button is located in the top-left corner of the grid. The right side of the screen features three code panels: HTML, CSS, and JS.

HTML

```
<p>Check <a href="https://codepen.io/matuzo/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: 2px 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** PRO

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;
  }
}
```

Console Assets Comments *

Last saved 6 months ago Delete Share Export Embed Collections

ANIMATION

bit.ly/grid_animation2

GridBugs

Inspired by [Flexbugs](#) this list aims to be a community curated list of CSS Grid Layout bugs, incomplete implementations and interop issues. [Grid shipped into browsers](#) in a highly interoperable state, however there are a few issues - let's document any we find here.

While I'd like to focus on issues relating to the [Grid Specification](#) here, it is likely that related specs such as Box Alignment will need to be referenced. If you think you have spotted an issue and it seems to relate to Grid, [post it](#). We can work out the details together and make sure browser or spec issues get logged in the right place.

Also, please raise an issue if I am wrong about any of these or you have more info or examples to add. Quite possibly I've pointed the finger at the wrong UA or missed a change in the spec. Help to make the list accurate appreciated!

This is not CSS Grid technical support

Please raise issues about interop issues where you have already been through the process of creating a [Reduced Test Case](#) to check that the issue isn't something in your code. If you want to learn about CSS Grid Layout then check out [Grid by Example](#) where I have video tutorials, small examples and links to other resources. I will answer more general grid questions on my [AMA](#) when time allows. I also have information regarding [fallbacks for older browsers](#).

The bugs

- [1. `grid-auto-rows` and `grid-auto-columns` should accept a track listing](#)
- [2. Repeat notation should accept a track listing](#)
- [3. Fragmentation support](#)
- [4. Sizing of items with an intrinsic size inside fixed size grid tracks](#)
- [5. Items with an intrinsic aspect ratio should align start](#)
- [6. The `grid-gap` property should accept percentage values](#)
- [7. Grid gaps should behave as auto-sized tracks?](#)
- [8. Setting max-height to a percentage should scale down a larger image inside a grid track](#)
- [9. `min-content` and `max-content` in track listings](#)
- [10. Some HTML elements can't be grid containers](#)
- [11. A textarea that is a grid item collapses to zero width](#)
- [12. Space distributed to empty tracks spanned by an item with content](#)
- [13. Inconsistency with percentage padding and margins](#)
- [14. `fit-content` is stretching](#)

1. `grid-auto-rows` and `grid-auto-columns` should accept a track listing

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

The properties `grid-auto-rows` and `grid-auto-columns` enable an author to set the size of tracks created in the implicit grid. The spec says:

"If multiple track sizes are given, the pattern is repeated as necessary to find the size of the implicit tracks. The first implicit grid track before the explicit grid receives the first specified size, and so on forwards; and the last implicit grid track before the explicit grid receives the last specified size, and so on backwards." - [7.6 Implicit Track Sizing](#)

Is there a grid system for CSS Grid Layout?

Is there a grid system for CSS Grid Layout?

- Grid is a grid system!

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.

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" by Manuel Matuzovic. The interface is divided into several sections:

- Grid**: A section containing four items: item 0 (green), item 3 (teal), item1 (red), and item2 (red).
- Flexbox**: A section containing four items: item 0 (green), item1 (red), item2 (red), and item 3 (teal).
- HTML**: The HTML code for both sections.
- CSS**: The CSS code for both sections.
- JS**: The JavaScript code for both sections.

The CSS code for the Grid section is as follows:

```
1<h2>Grid</h2>
2<section class="grid">
3  <div class="item">item1</div>
4  <div class="item">item2</div>
5</section>
6
7<h2>Flexbox</h2>
8<section class="flex">
9  <div class="item">item1</div>
10 <div class="item">item2</div>
11</section>
12
13
```

The CSS code for the Flexbox section is as follows:

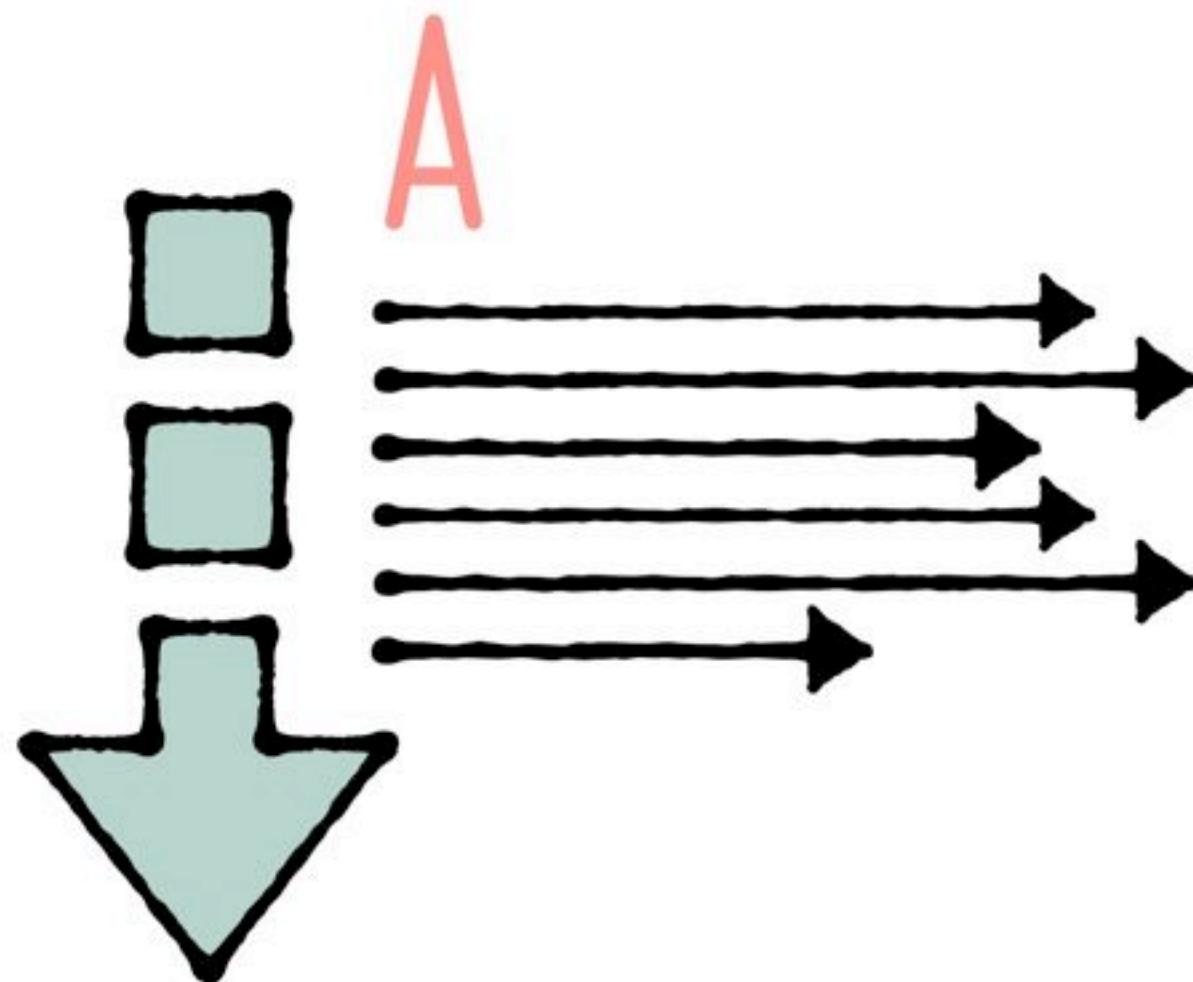
```
1<.grid> {
2  display: grid;
3  grid-template-columns: 1fr 1fr 1fr
4  1fr;
5  grid-template-rows: 100px 100px;
6  grid-gap: 20px;
7}
8<.flex> {
9  display: flex;
10 justify-content: space-between;
11}
13<section>:before {
14  content: "item 0";
15  display: block;
16  background-color: #7CCBB0;
17  border-radius: 5px;
18  padding: 10px;
19}
22<section>:after {
23  content: "item 3";
24  display: block;
25  grid-column: 2;
26  grid-row: 1 / span 2;
27  background-color: #7CCBB0;
28  border-radius: 5px;
29  padding: 10px;
30}
31
32
33
```

PSEUDO ELEMENTS

bit.ly/grid_pseudo



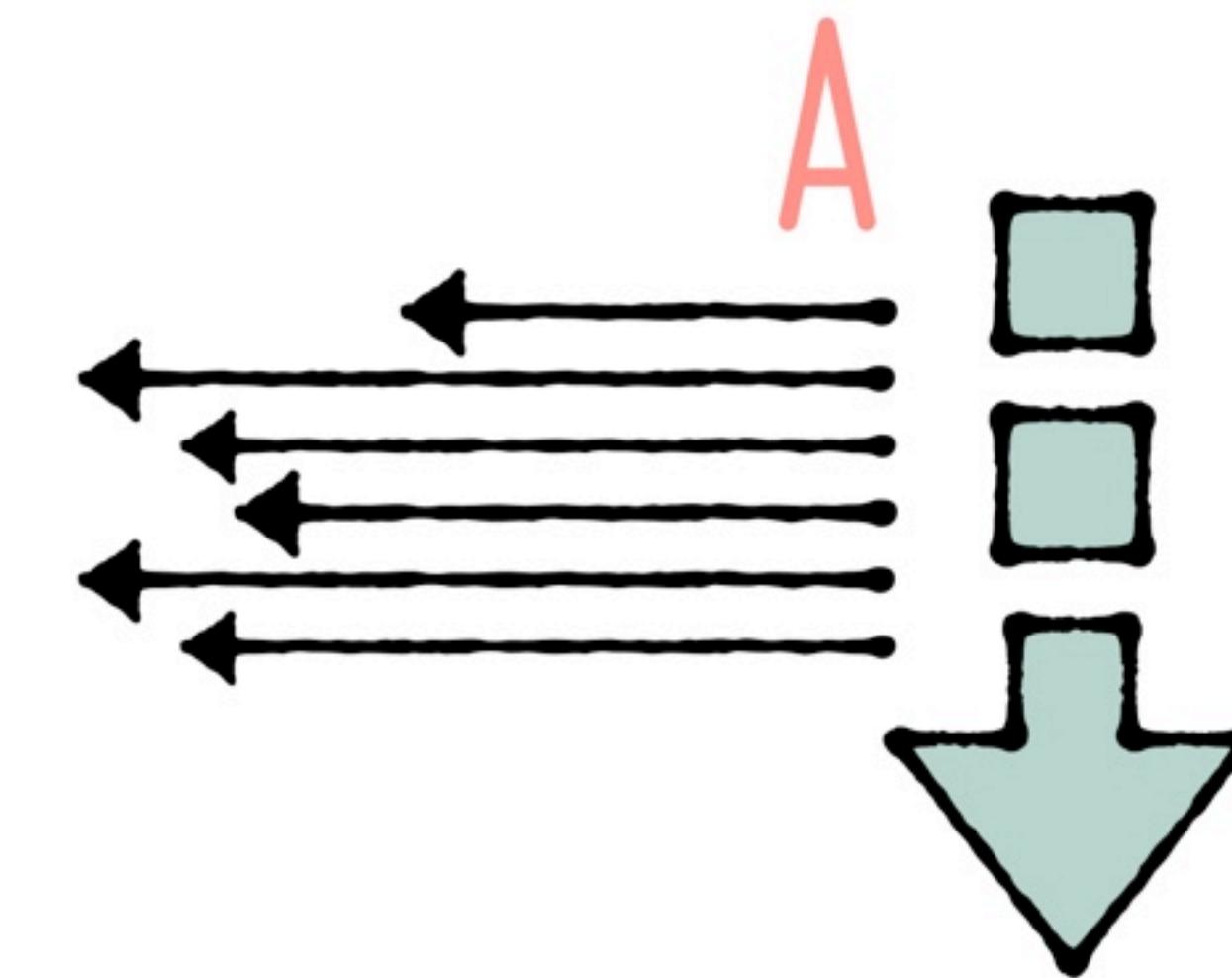
LATIN-BASED SYSTEMS





WRITING MODE: ARABIC-BASED SYSTEMS

ARABIC-BASED SYSTEMS



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>
```

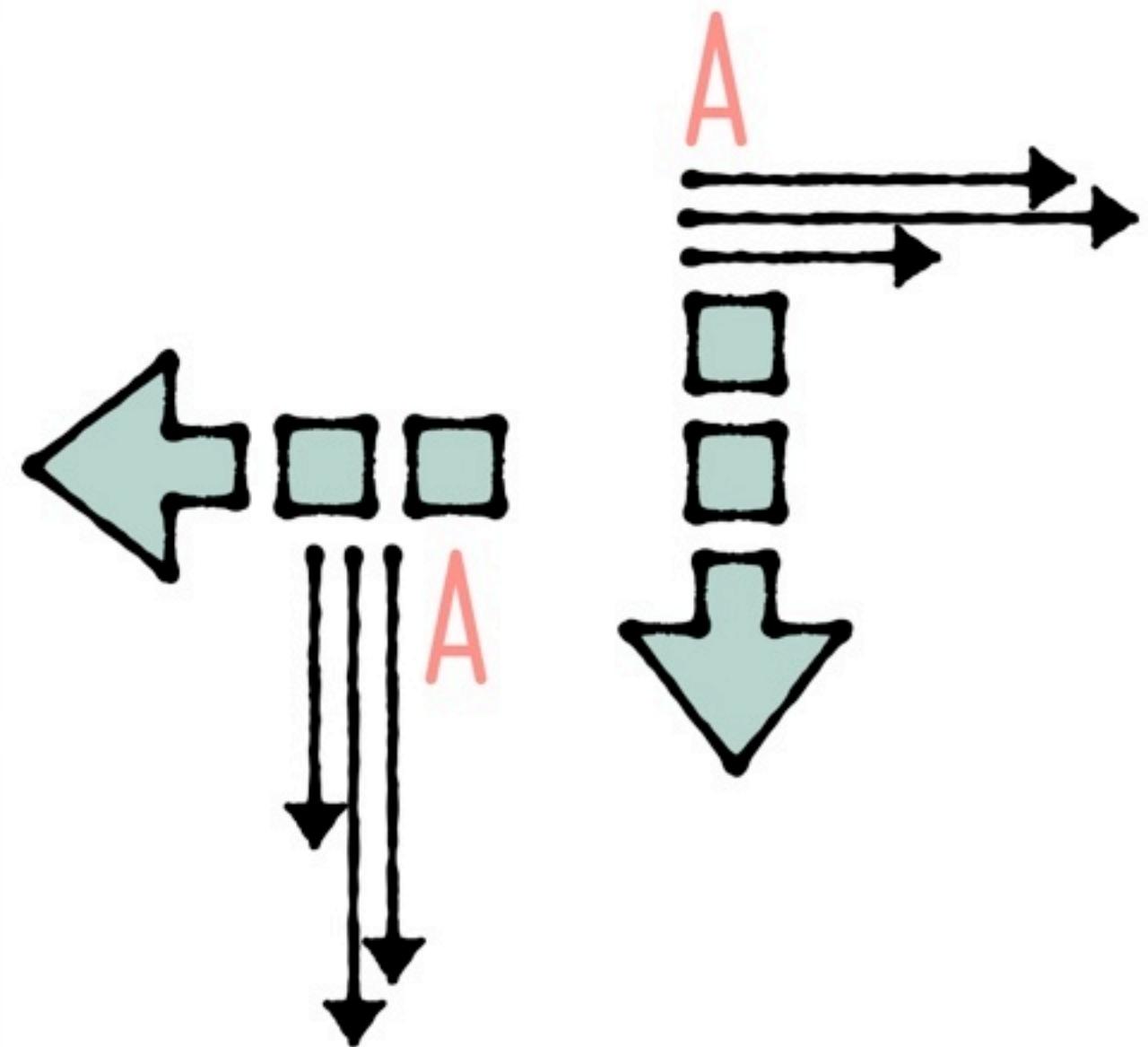
CSS

```
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



WRITING MODE: HAN-BASED SYSTEMS

24ways.org/2016/css-writing-modes

A screenshot of a CodePen editor window displaying a CSS Grid layout example. The page title is "CSS Grid Layout: vertical-rl writing-mode". The layout consists of five red rectangular elements arranged in a grid pattern within a red-bordered container. The elements are labeled "Element 1" through "Element 5". The CSS code includes rules for the grid container and individual items, specifically targeting the second item with `justify-self: start;`.

```
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>
</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;
}
```

JS

A screenshot of a CodePen editor window titled "CSS Grid Layout: Anonymous items". The URL is <https://codepen.io/matuzo/pen/GQKpPz?editors=1100>. The title bar shows "CSS Grid Layout: Anonymous items" and "A PEN BY Manuel Matuzovic PRO". The main content area displays the text "CSS Grid Layout is awesome." in a large, bold, sans-serif font. The "HTML" panel contains the following code:

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

The "CSS" panel contains the following styles:

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

The "JS" panel is empty. At the bottom, there are tabs for "Console", "Assets", "Comments", and "⌘", along with buttons for "Delete", "Share", "Export", "Embed", and "Collections". A status message says "Last saved less than a minute ago".

ANONYMOUS ITEMS

bit.ly/grid_anon

WHAT'S NEXT?

Nesting, Debugging, RWD

Gaps

Gaps

- `grid-column-gap`, `grid-row-gap` and `grid-gap` will be renamed to `column-gap`, `row-gap` and `a gap`.

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.

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.

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

Subgrid

- Only direct children of a grid container become grid items. Children of grid items are no affected.

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

- 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

display:contents

- The element itself does not generate any boxes, but its children and pseudo-elements still generate boxes as normal.

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.

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 CodePen editor interface with a dark theme. The top bar includes a title "display: contents", a URL "https://codepen.io/matuzo/pen/EoMbLJ?editors=1100", and various navigation and settings buttons. The main area is divided into sections: a preview window on the left showing a red-bordered box with text, and three code editors on the right: HTML, CSS, and JS.

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 #D73B38;
  border-radius: 5px;
  background-color: #D73B38;
  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!");
```

The preview window displays two paragraphs of text. The first paragraph is enclosed in a red border, while the second is in a grey border. The CSS section shows the styling for these elements, with the red border applied to the inner div and the grey border to the outer content div. The JS section contains a simple log statement.

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+ <h1>Fake subgrid</h1>
15+ <div class="wrapper subgrid">
16+   <h2 class="item">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+
25+   </ul>
26+ </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+ }
```

JS

DISPLAY: CONTENTS

bit.ly/grid_contents1

Subgrid with display:contents?

Subgrid with display:contents?

- Issue 1: You cannot apply backgrounds and borders to an item with display: contents.

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.

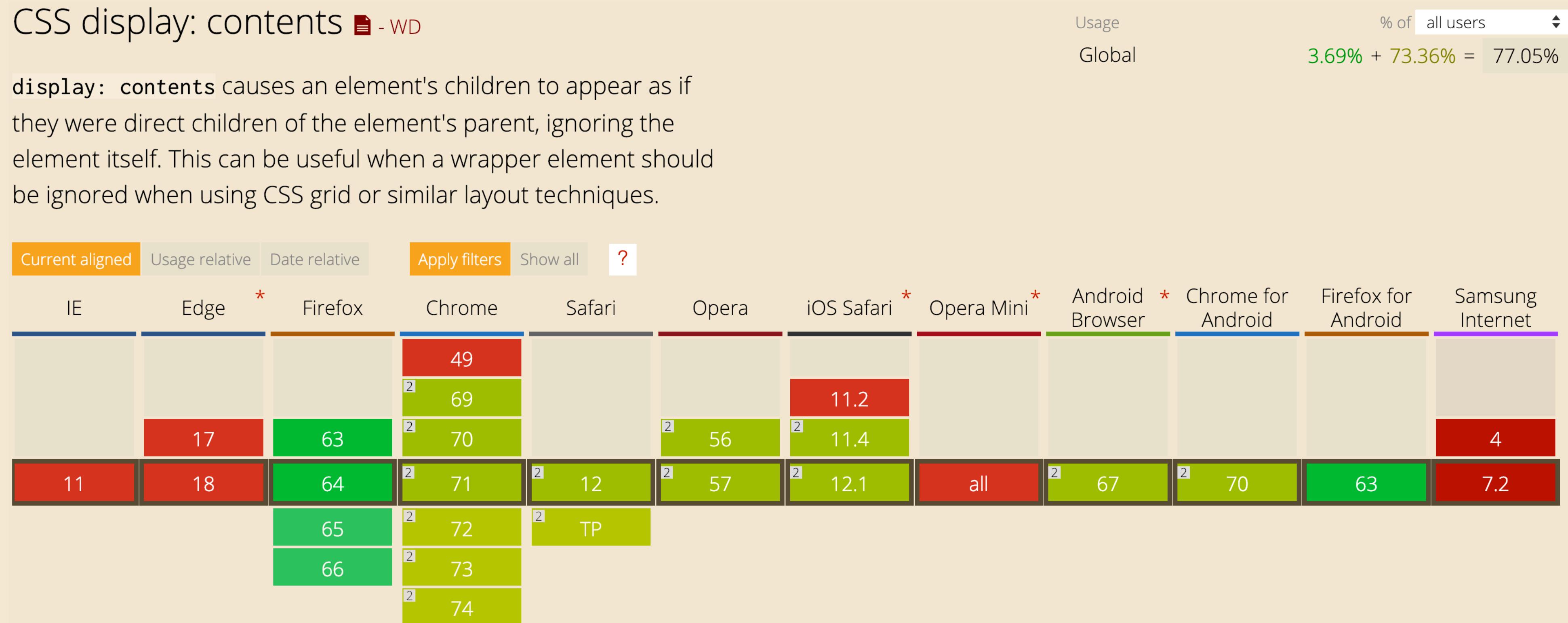
Subgrid with display:contents?

- 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.



Masonry Layout ???

Masonry Layout ???

- Grid was designed for the creation of two dimensional grids, which a Masonry Layout isn't.

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.

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 cells, tracks and areas

- Styling lines, cells, tracks and areas is currently not possible.

Styling cells, tracks and areas

- Styling lines, cells, tracks and areas is currently not possible.
- It would be nice, though.

Styling cells, tracks and areas

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

More

More

- Creating non-rectangular grid areas.

More

- Creating non-rectangular grid areas.
- Flowing content through grid cells or areas.

More

- Creating non-rectangular grid areas.
- Flowing content through grid cells or areas.
- Solving the keyboard/layout disconnect.

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

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.

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.

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.

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.

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

Wrap-UP

- Explicit grids. (grid-template-rows and grid-template-columns)

Wrap-UP

- Explicit grids. (grid-template-rows and grid-template-columns)
- Implicit grids (grid-auto-rows and grid-auto-columns)

Wrap-UP

- Explicit grids. (grid-template-rows and grid-template-columns)
- Implicit grids (grid-auto-rows and grid-auto-columns)
- Flow (column, row, dense)

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)

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

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)

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

Wrap-UP

- **Box Alignment Module** (`justify-items`, `align-items`, `justify-self`, `align-self`, `justify-content`, `align-content`, `grid-gap` (`gap`)).

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.

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.

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/>