A guide to learning CSS grid by @jonsuh

CSS Grid is a powerful tool that allows for two-dimensional layouts to be created on the web. This guide was created as a resource to help you better understand and learn Grid, and was organized in a way I thought made the most sense when learning it.

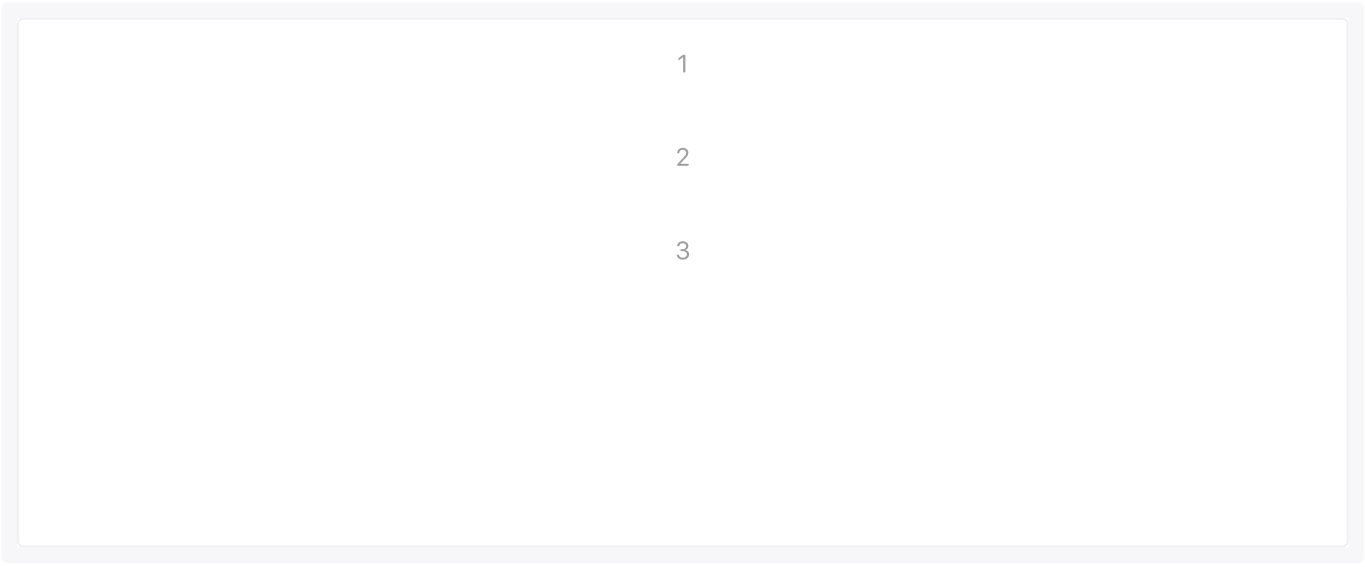
Table of Contents

Grid Container

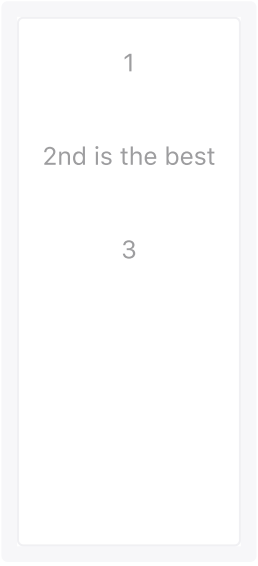
Create a grid container by setting the `display` property with a value of `grid` or `inline-grid`. All direct children of grid containers become grid items.

`display: grid`

Grid items are placed in rows by default and span the full width of the grid container.



`display: inline-grid`



Explicit Grid

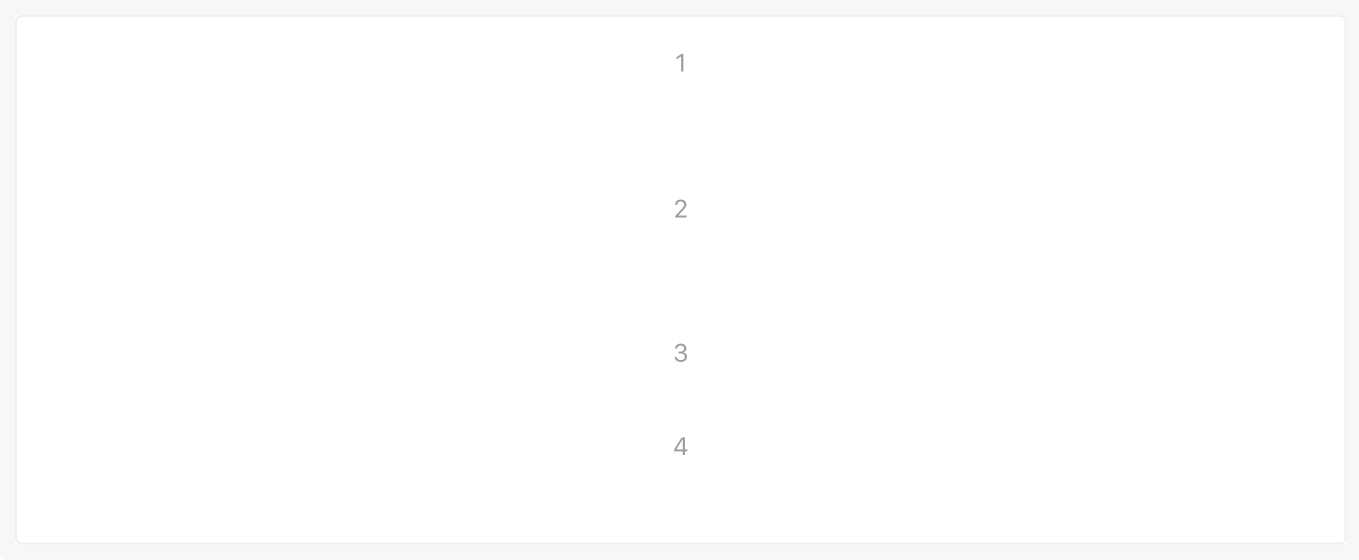
Explicitly set a grid by creating columns and rows with the `grid-template-columns` and `grid-template-rows` properties.

```
grid-template-rows: 50px 100px
```

A row track is created for each value specified for `grid-template-rows`. Track size values can be any non-negative, length value (`px`, `%`, `em`, etc.)

Items 1 and 2 have fixed heights of `50px` and `100px`.

Because only 2 row tracks were defined, heights of items 3 and 4 are defined by the contents of each.

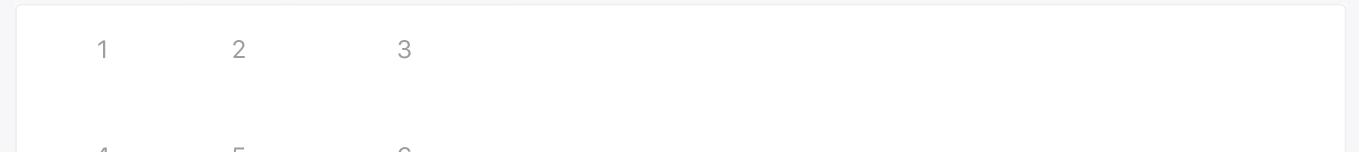



```
grid-template-columns: 90px 50px 120px
```

Like rows, a column track is created for each value specified for `grid-template-columns`.

Items 4, 5 and 6 were placed on a new row track because only 3 column track sizes were defined; and because they were placed in column tracks 1, 2 and 3, their column sizes are equal to items 1, 2 and 3.

Grid items 1, 2 and 3 have fixed widths of `90px`, `50px` and `120px` respectively.



4 5 6

```
grid-template-columns: 1fr 1fr 2fr
```

The `fr` unit helps create flexible grid tracks. It represents a fraction of the available space in the grid container (works like Flexbox's unitless values).

In this example, items 1 and 2 take up the first two (of four) sections while item 3 takes up the last two.

1 2 3

```
grid-template-columns: 3rem 25% 1fr 2fr
```

`fr` is calculated based on the remaining space when combined with other length values.

In this example, `3rem` and `25%` would be subtracted from the available space before the size of `fr` is calculated:
`1fr = ((width of grid) - (3rem) - (25% of width of grid)) / 3`

1 2 3 4

Minimum and Maximum Grid Track Sizes

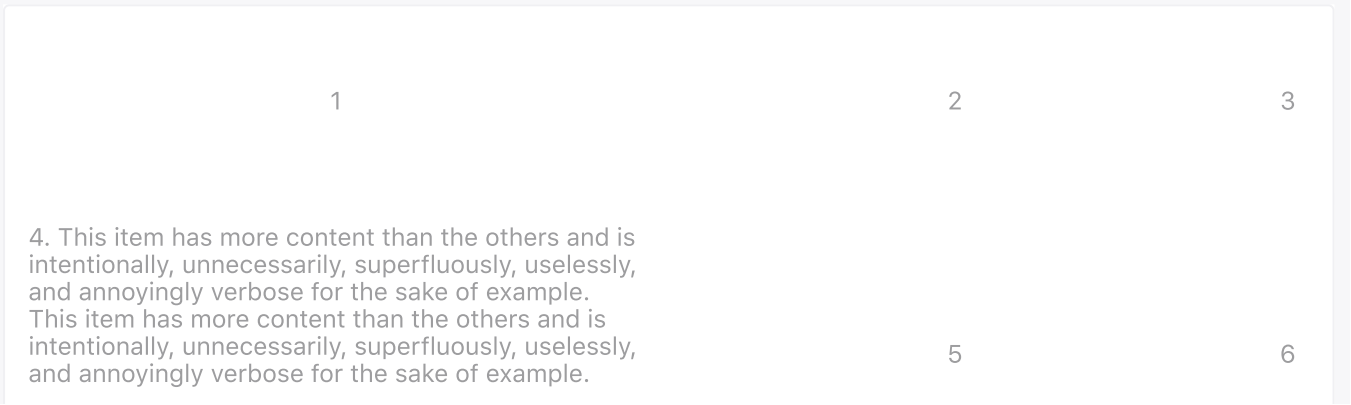
Tracks sizes can be defined to have a minimum and/or maximum size with the `minmax()` function.

```
grid-template-rows:    minmax(100px, auto);
grid-template-columns: minmax(auto, 50%) 1fr 3em;
```

The `minmax()` function accepts 2 arguments: the first is the minimum size of the track and the second the maximum size. Alongside length values, the values can also be `auto`, which allows the track to grow/stretch based on the size of the content.

In this example, the first row track is set to have a minimum height of `100px`, but its maximum size of `auto` will allow the row track to grow it the content is taller than `100px`.

The first column track has a minimum size of `auto`, but its maximum size of `50%` will prevent it from getting no wider than `50%` of the grid container width.



This item has more content than the others and is intentionally, unnecessarily, superfluously, uselessly, and annoyingly verbose for the sake of example.

Repeating Grid Tracks

Define repeating grid tracks using the `repeat()` notation. This is useful for grids with items with equal sizes or many items.

```
grid-template-rows:    repeat(4, 100px);
grid-template-columns: repeat(3, 1fr);
```

The `repeat()` notation accepts 2 arguments: the first represents the number of times the defined tracks should repeat, and the second is the track definition.

1 2 3

4 5 6

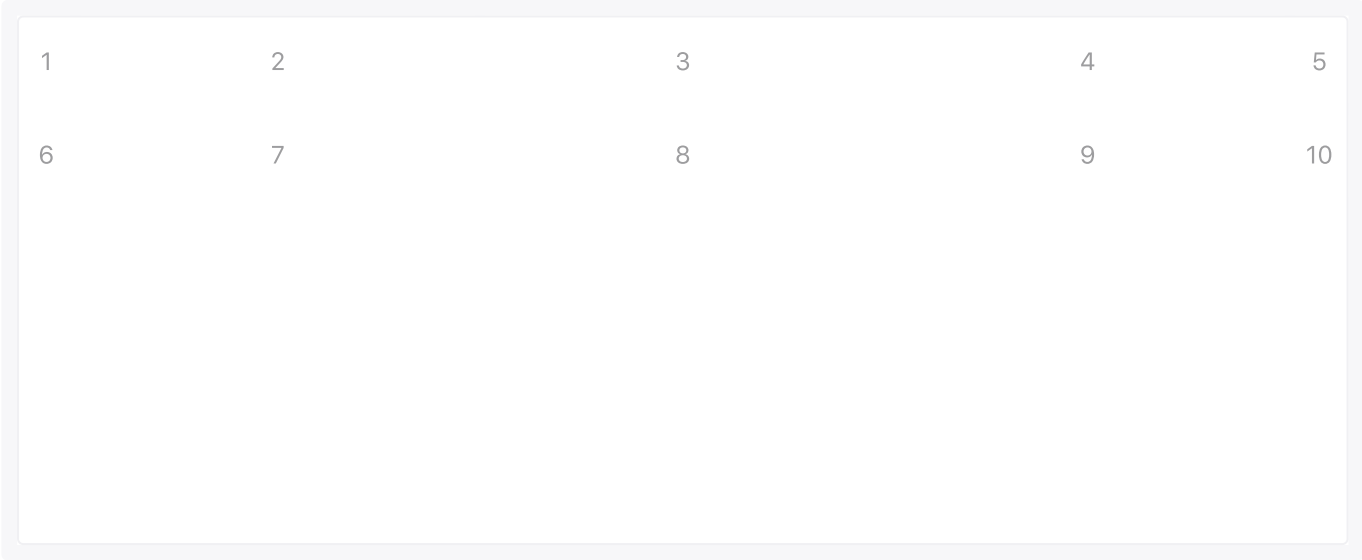
7 8 9

10 11 12

```
grid-template-columns: 30px repeat(3, 1fr) 30px
```

`repeat()` can also be used within track listings.

In this example, the first and last column tracks have widths of `30px`, and the 3 column tracks in between, created by `repeat()`, have widths of `1fr` each.



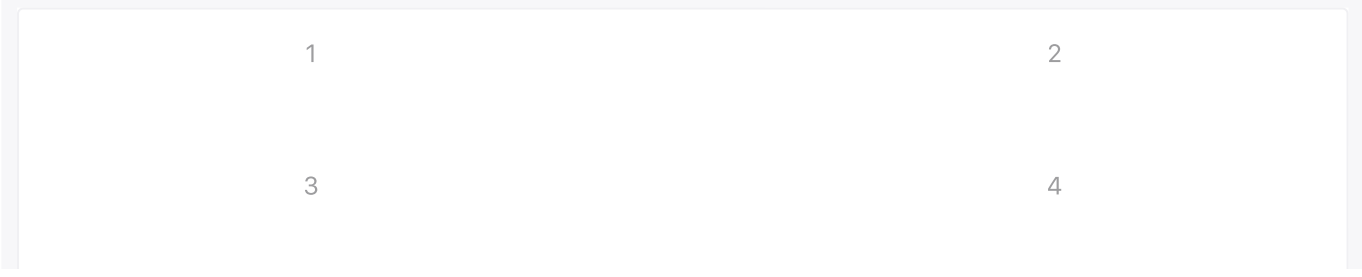
Grid Gaps (Gutters)

The `grid-column-gap` and `grid-row-gap` properties create gutters between columns and rows.

Grid gaps are only created in between columns and rows, and not along the edge of the grid container .

```
grid-row-gap:    20px;
grid-column-gap: 5rem;
```

Gap size values can be any non-negative, length value (`px`, `%`, `em`, etc.)



```
grid-gap: 100px 1em
```

`grid-gap` is shorthand for `grid-row-gap` and `grid-column-gap`.

If two values are specified, the first represents `grid-row-gap` and the second `grid-column-gap`.


```
grid-gap: 2rem
```

One value sets equal row and column gaps.

Positioning Items by Grid Line Numbers

Grid lines are essentially lines that represent the start of, the end of, or between column and row tracks.

Each line, starting from the start of the track and in the direction of the grid, is numbered incrementally starting from 1.

```
grid-row-start:    2;
grid-row-end:      3;
grid-column-start: 2;
grid-column-end:   3;
```

This 2-column by 3-row grid results in 3 column lines and 4 row lines. Item 1 was repositioned by row and column line numbers.

If an item spans only one row or column, `grid-row/column-end` is not necessary.

```
grid-row:    2;
grid-column: 3 / 4;
```

`grid-row` is shorthand for `grid-row-start` and `grid-row-end`.

`grid-column` is shorthand for `grid-column-start` and `grid-column-end`.

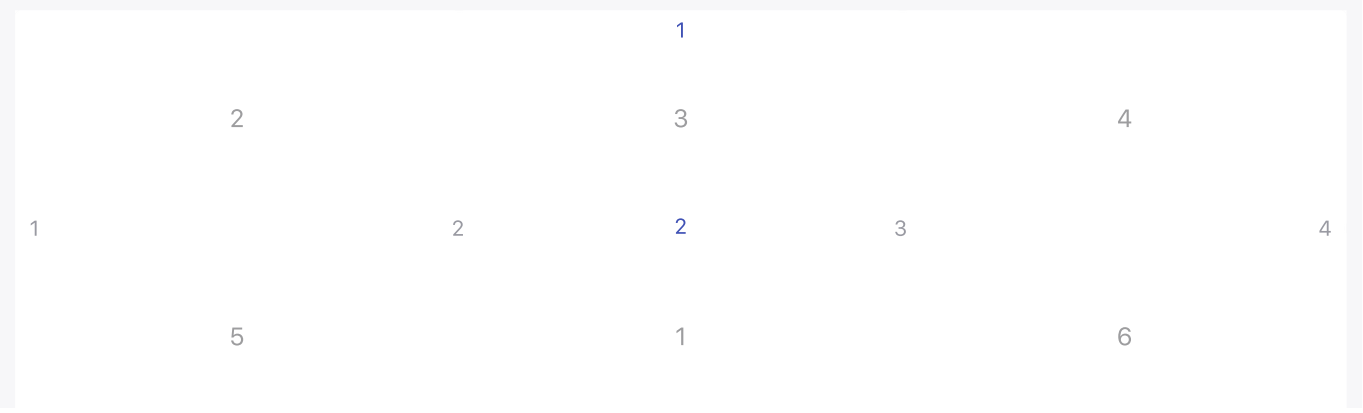
If one value is provided, it specifies `grid-row/column-start`.

If two values are specified, the first value corresponds to `grid-row/column-start` and the second `grid-row/column-end`, and must be separated by a forward slash `/`.


```
grid-area: 2 / 2 / 3 / 3
```

`grid-area` is shorthand for `grid-row-start`, `grid-column-start`, `grid-row-end` and `grid-column-end`.

If four values are specified, the first corresponds to `grid-row-start`, the second `grid-column-start`, the third `grid-row-end` and the fourth `grid-column-end`.



3

Spanning Items Across Rows and Columns

Grid items span only one column and row track by default, but can span multiple row and/or column tracks using the same properties to position them.

```
grid-column-start: 1;
grid-column-end:   4;
```

Set a grid item to span more than one column track by setting `grid-column-end` to a column line number that is more than one column away from `grid-column-start`.

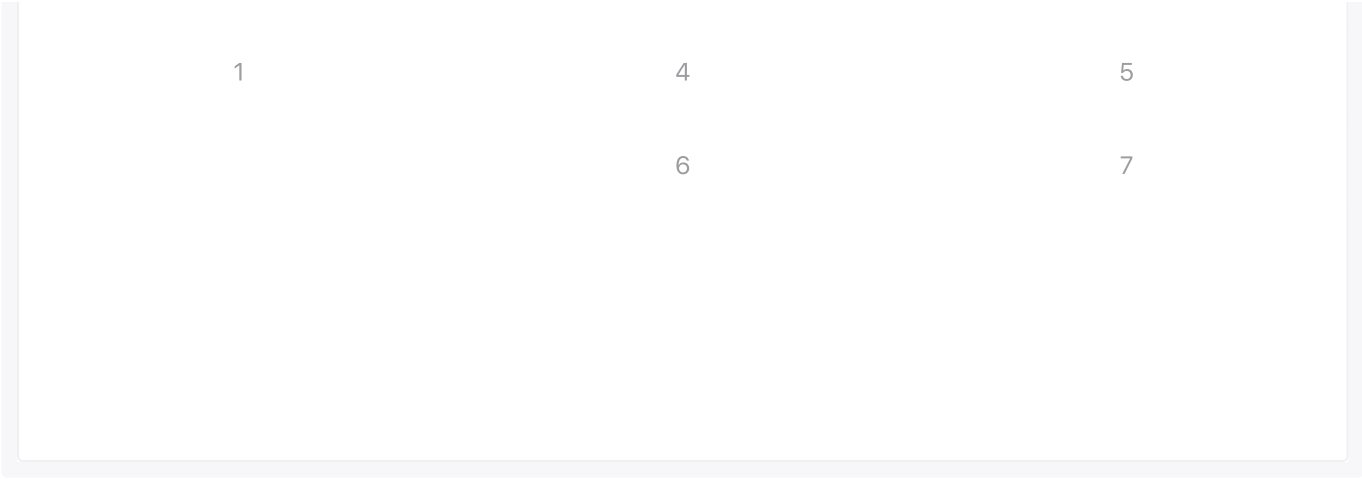
1

2 3 4

```
grid-row-start: 1;
grid-row-end:   4;
```

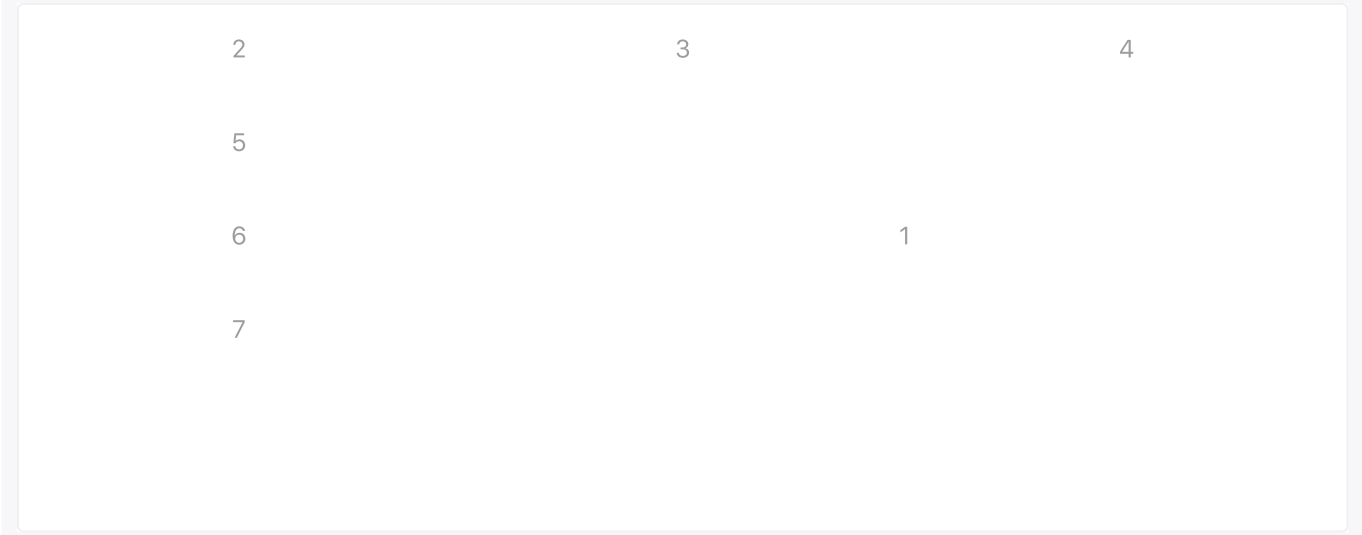
Grid items can also span across multiple row tracks by setting `grid-row-end` to more than one row track away.

2 3



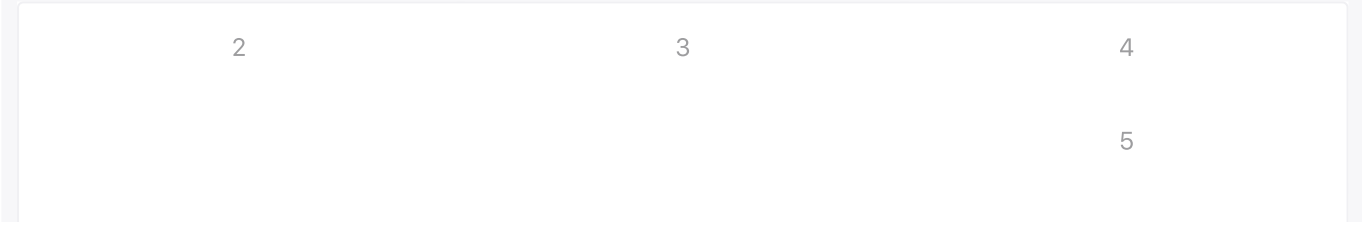

```
grid-row:    2 / 5;
grid-column: 2 / 4;
```

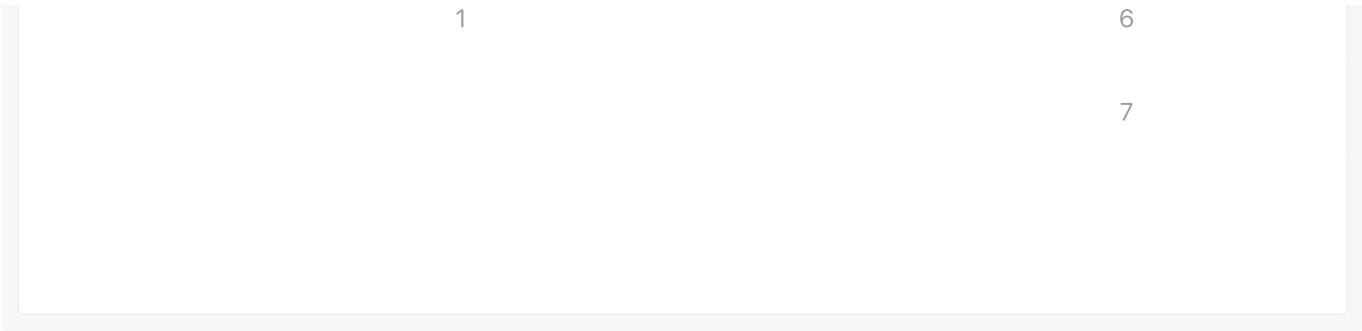
Shorthand properties `grid-row` and `grid-column` can also be used to position and span grid items more than one row or column.




```
grid-row:    2 / span 3;
grid-column: span 2;
```

The keyword `span`, followed by the # of columns or rows to span, can also be used.





Naming Grid Lines

Grid lines can be named when defining the grid with the `grid-template-rows` and `grid-template-columns` properties. Line names can then be referenced to position grid items.

```
grid-template-rows:    [row-1-start] 1fr [row-2-start] 1fr [row-2-end];
grid-template-columns: [col-1-start] 1fr [col-2-start] 1fr [col-3-start] 1fr [c
```

Assign names to grid lines when defining your grid with the `grid-template-rows` and `grid-template-columns` properties.

In line names, avoid keywords that appear in the specification (e.g. `span`) to not cause confusion.

Assigned line names must be wrapped in square brackets `[name-of-line]` and placed relative to the grid tracks.

```
grid-template-rows:    [row-start row-1-start] 1fr [row-1-end row-2-start] 1fr
grid-template-columns: [col-start] 1fr [col-2-start] 1fr [col-3-start] 1fr [col
```

Multiple names can be assigned to grid lines by adding names within square brackets and separating each with a whitespace.

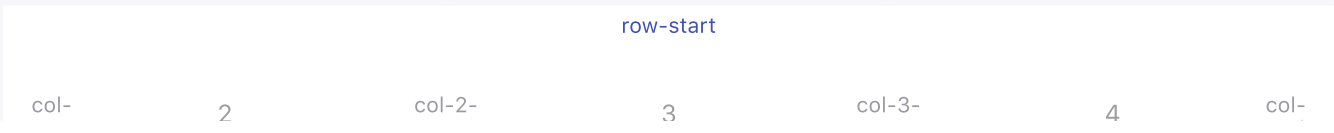
Each line name can then be referenced when positioning grid items by line names.

Positioning Items by Line Names

With named grid lines, items can be positioned by line names and numbers.

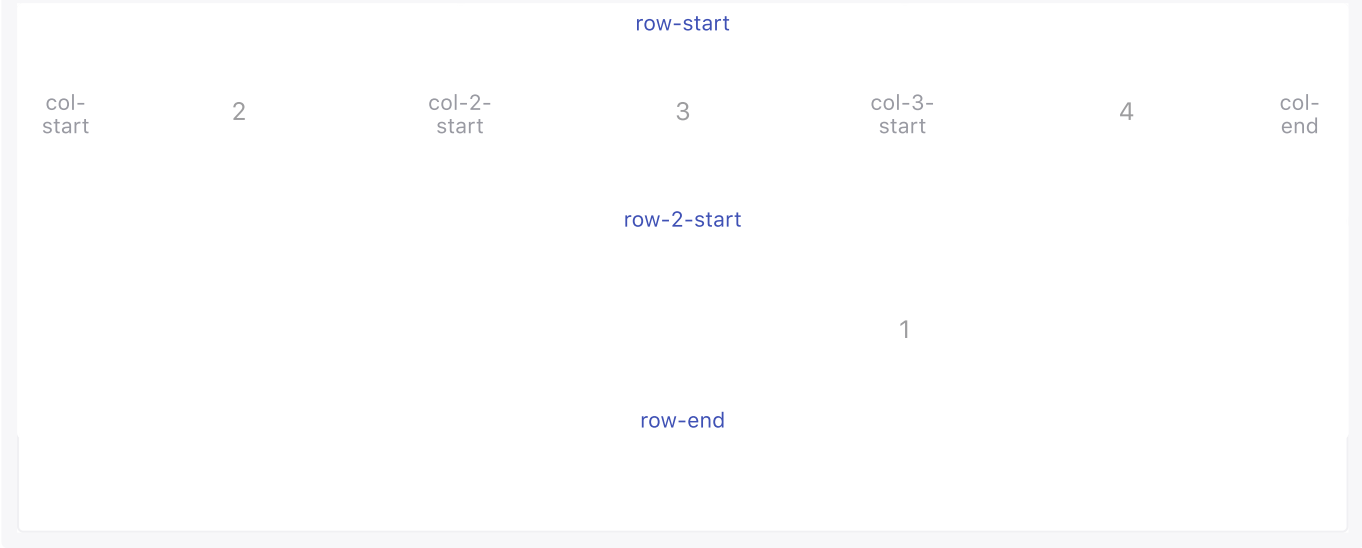
```
grid-row-start:    row-2-start;
grid-row-end:      row-end;
grid-column-start: col-2-start;
grid-column-end:   col-end;
```

Referenced line names should not be wrapped in square brackets.




```
grid-row:    row-2-start / row-end;
grid-column: col-2-start / col-end;
```

`grid-row` and `grid-column` shorthand properties also support the use of grid line names when positioning items.



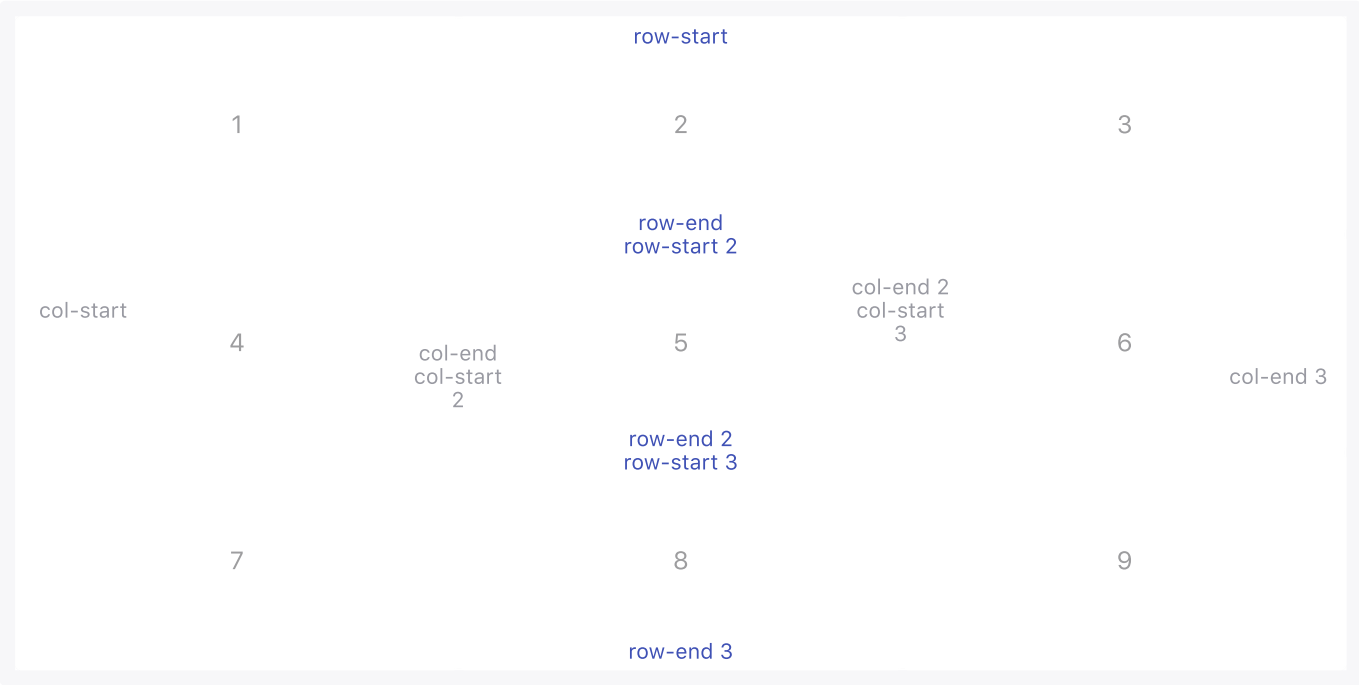
Naming and Positioning Items by Grid Lines with the Same Name

Lines can be assigned the same name with the `repeat()` function. This can save you time from having to name each line in track definitions.

```
grid-template-rows: repeat(3, [row-start] 1fr [row-end]);
grid-template-columns: repeat(3, [col-start] 1fr [col-end]);
```

Line name assignments can also be included within the `repeat()` function. This results in multiple grid lines with the same names.

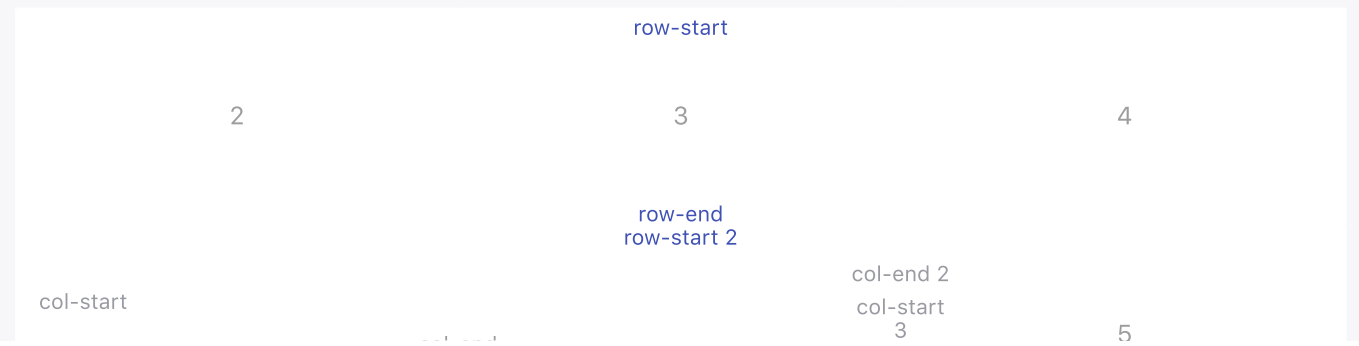
Lines with the same name are also assigned the a line's position/name's occurrence number, which allows it to be uniquely identified from another line with the same name.

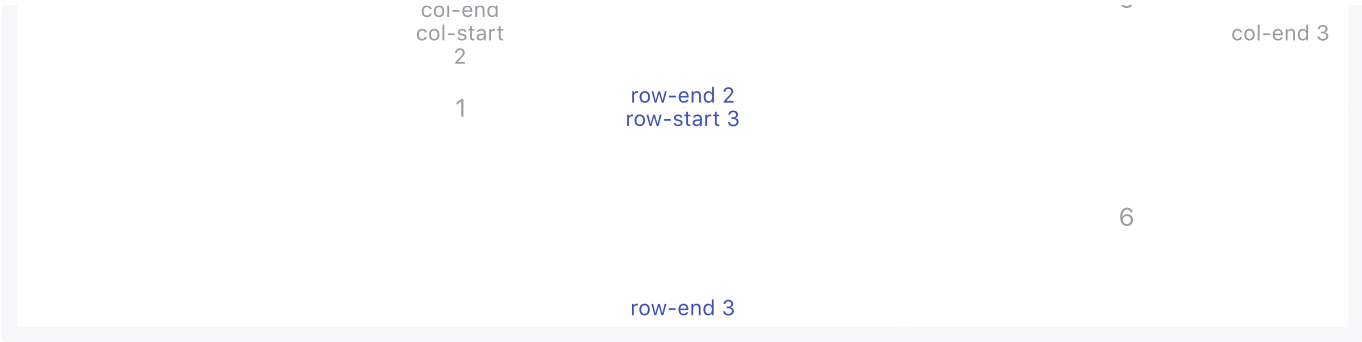



```
grid-row:    row-start 2 / row-end 3;
grid-column: col-start / col-start 3;
```

To position items by lines with the same name, reference the line's name and position/name's occurrence number—the name and number should be separated by a whitespace.

In this example, item 1's row position starts at the 2nd grid line named `row-start` and ends at the 3rd grid line named `row-end`; and its column position starts at the 1st grid line named `col-start` and ends at the 3rd grid line named `col-start`.





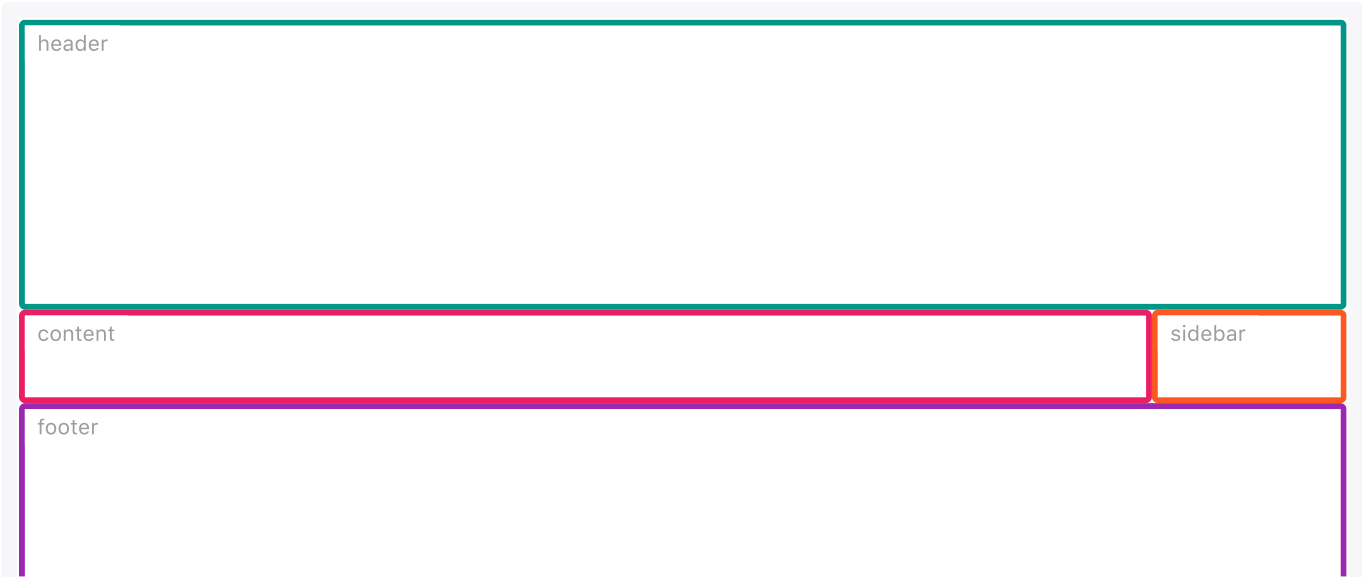
Naming and Positioning Items by Grid Areas

Like grid line names, grid areas can also be named with the `grid-template-areas` property. Names can then be referenced to position grid items.

```
grid-template-areas:   "header header"
                       "content sidebar"
                       "footer footer";
grid-template-rows:    150px 1fr 100px;
grid-template-columns: 1fr 200px;
```

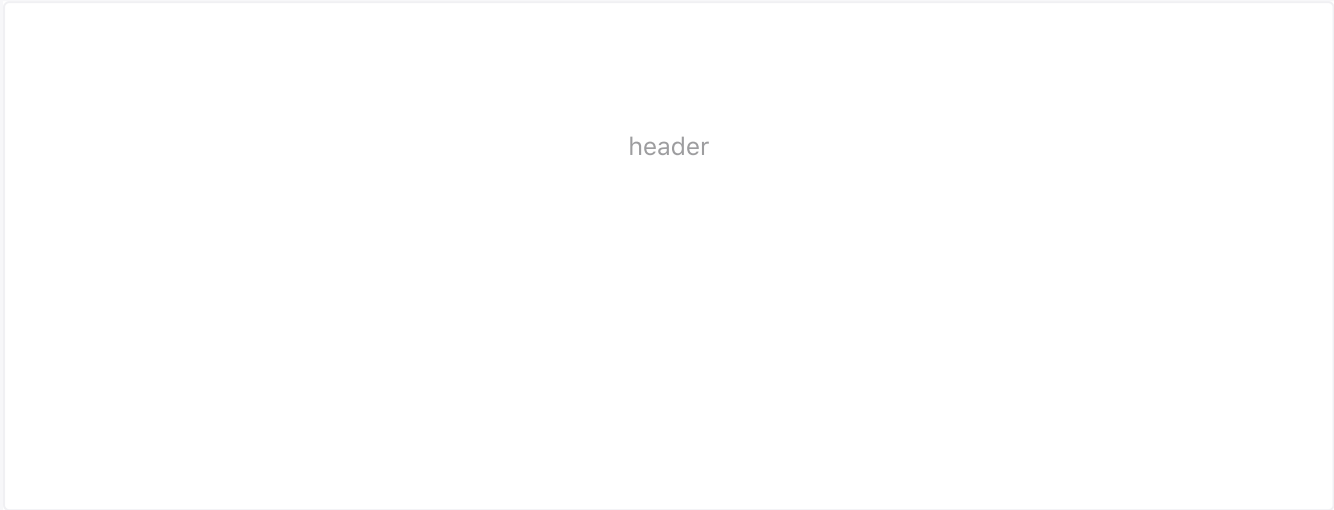
Sets of names should be surrounded in single or double quotes, and each name separated by a whitespace.

Each set of names defines a row, and each name defines a column.



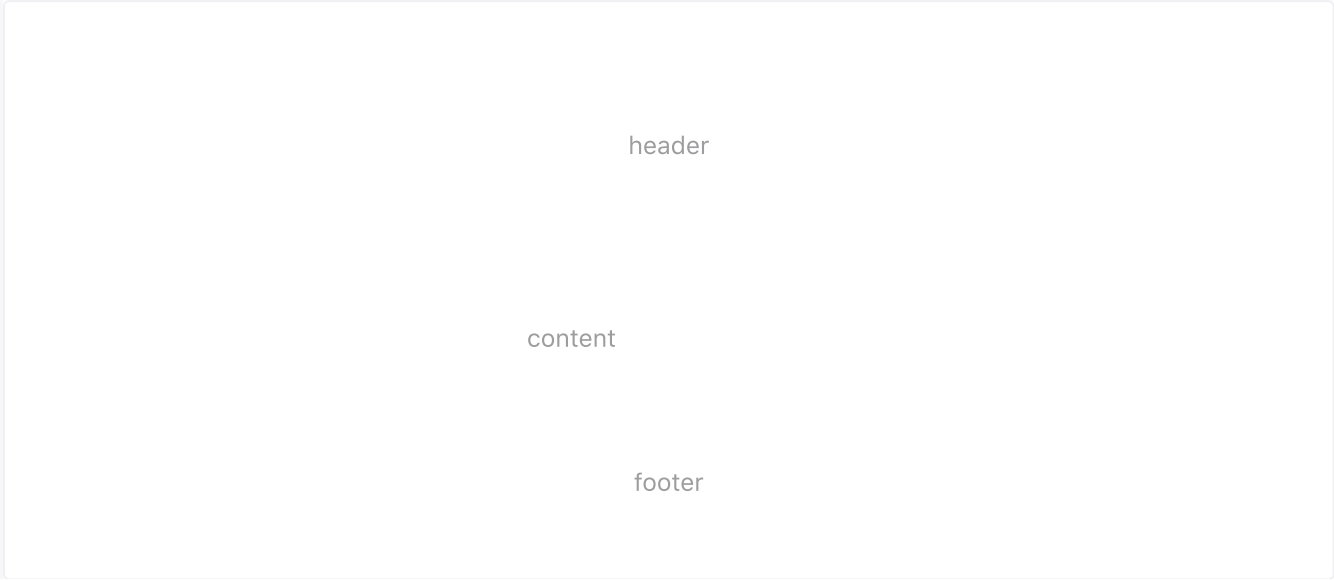
```
grid-row-start:    header;
grid-row-end:      header;
grid-column-start: header;
grid-column-end:   header;
```

Grid area names can be referenced by the same properties to position grid items: `grid-row-start`, `grid-row-end`, `grid-column-start`, and `grid-column-end`.



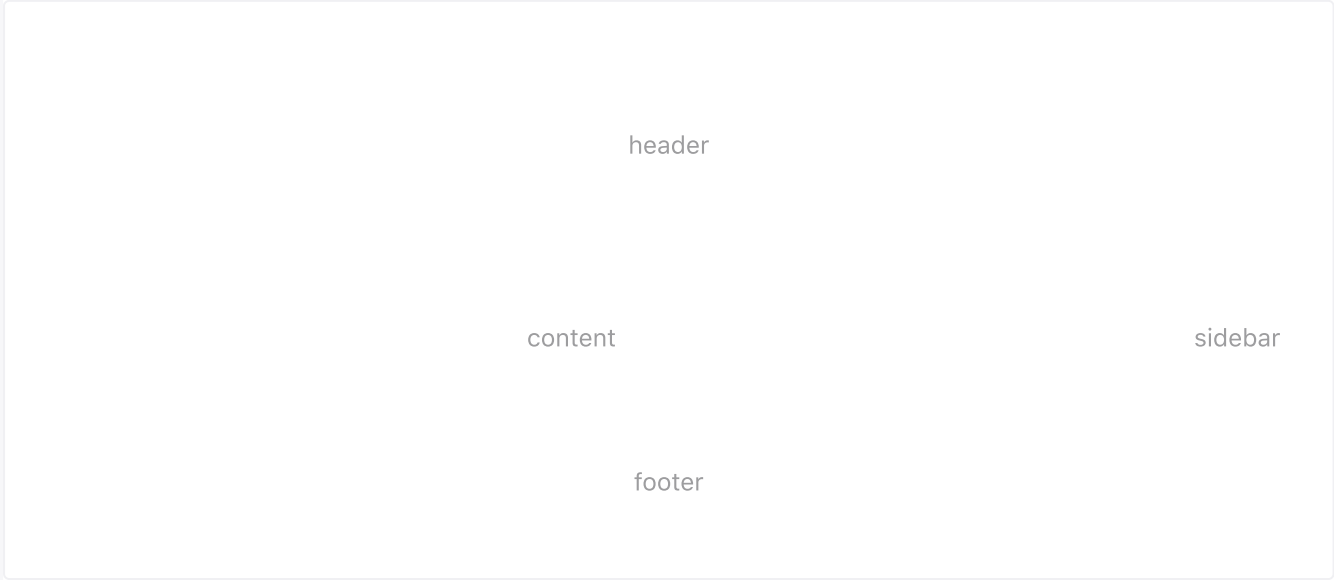

```
grid-row:    footer;
grid-column: footer;
```

The `grid-row` and `grid-column` shorthand properties can also reference grid area names.



```
grid-area: sidebar;
```

The `grid-area` shorthand property can also be used to reference grid area names.



Implicit Grid

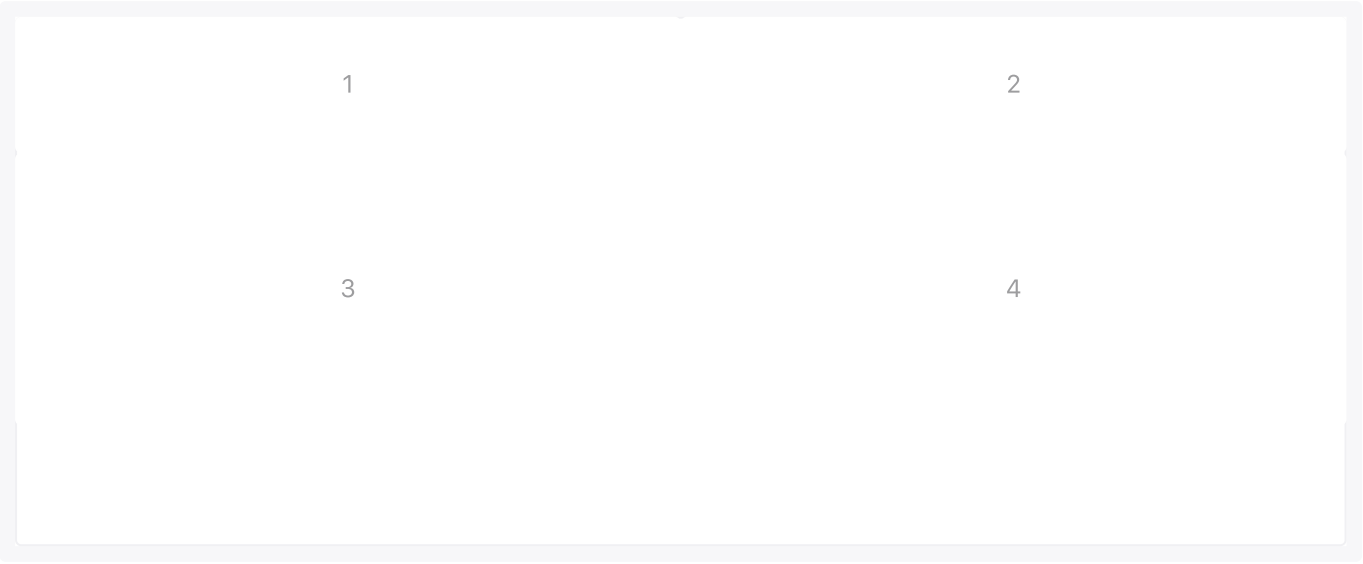
An implicit grid is created when a grid needs to position items outside of the explicit grid because there isn't enough space for items in the explicitly defined tracks or you decide to position something outside of the explicit grid. Those items are then auto-placed in the implicit grid.

The implicit grid can be defined using the `grid-auto-rows`, `grid-auto-columns`, and `grid-auto-flow` properties.

```
grid-template-rows:    70px;
grid-template-columns: repeat(2, 1fr);
grid-auto-rows:        140px;
```

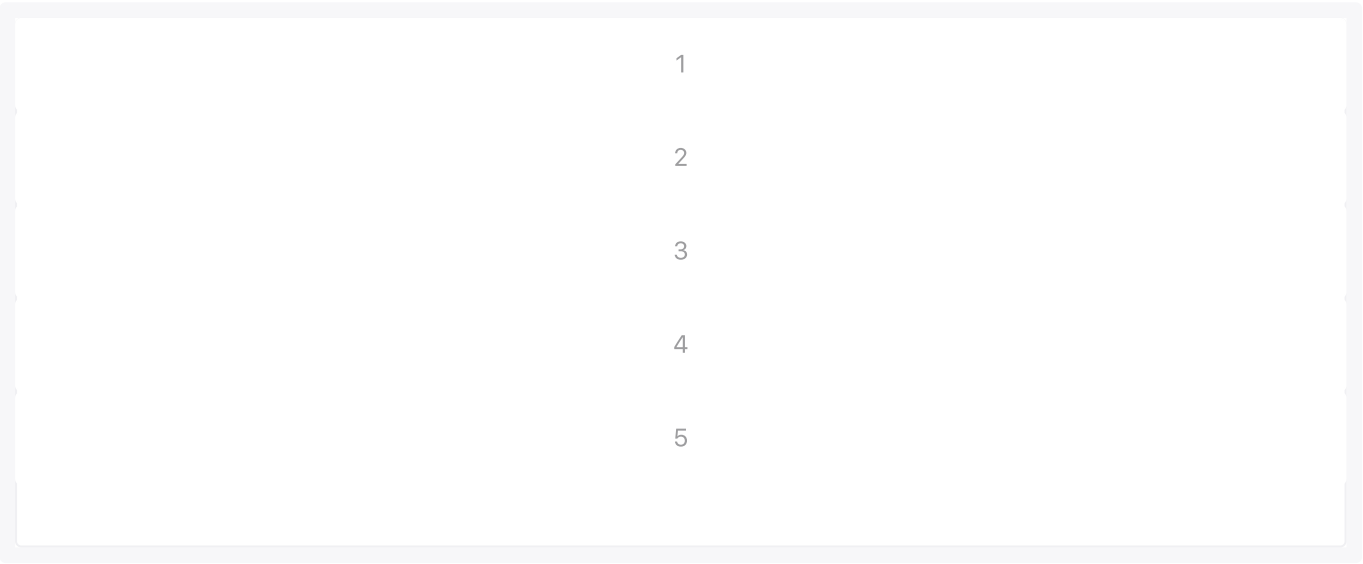
In this example we've only defined one row track, therefore grid items 1 and 2 are `70px` tall.

A second row track was auto-created to make room for items 3 and 4. `grid-auto-rows` defines the row track sizes in the implicit grid, which is reflected by the the `140px` heights of items 3 and 4.



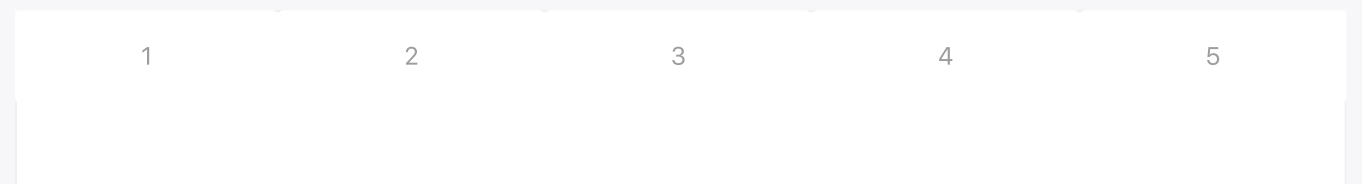
`grid-auto-flow: row`

The default flow (direction) of a grid is `row`.



`grid-auto-flow: column`

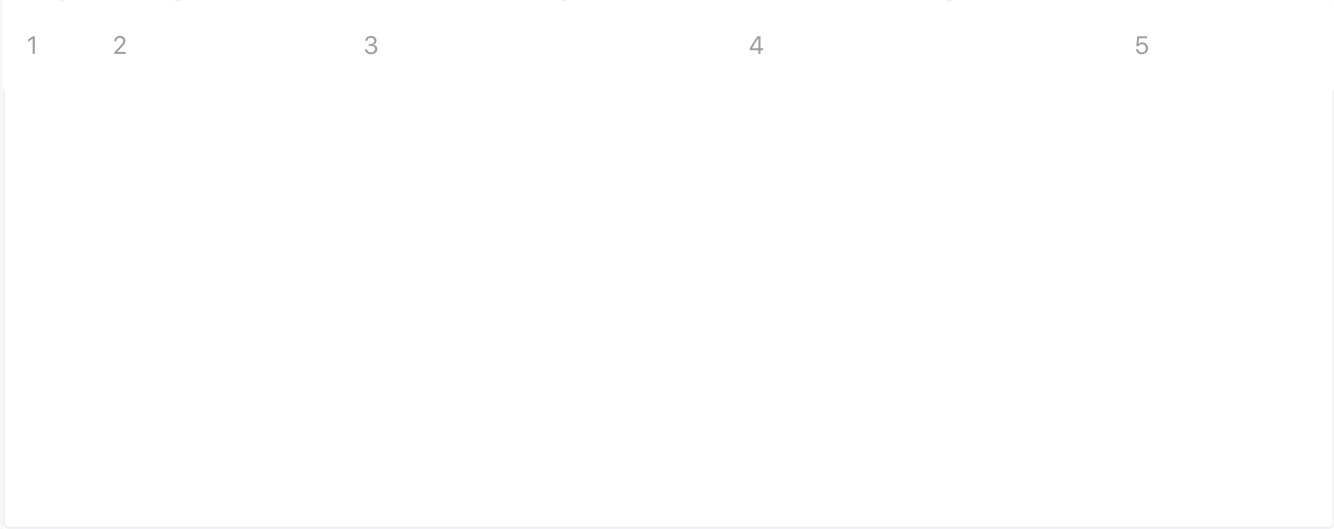
A grid's flow can be changed to `column`.



```
grid-template-columns: 30px 60px;
grid-auto-flow:        column;
grid-auto-columns:     1fr;
```

In this example, we've only defined the sizes of the first two column tracks—item 1 is `30px` wide and item 2, `60px`.

Column tracks are auto-created in the implicit grid to make room for items 3, 4 and 5; and track sizes are defined by `grid-auto-columns`.



Implicitly Named Grid Areas

Grid lines can generally be named whatever you'd like, but assigning names ending in `-start` and `-end comes` with added benefits—they implicitly create named grid areas,

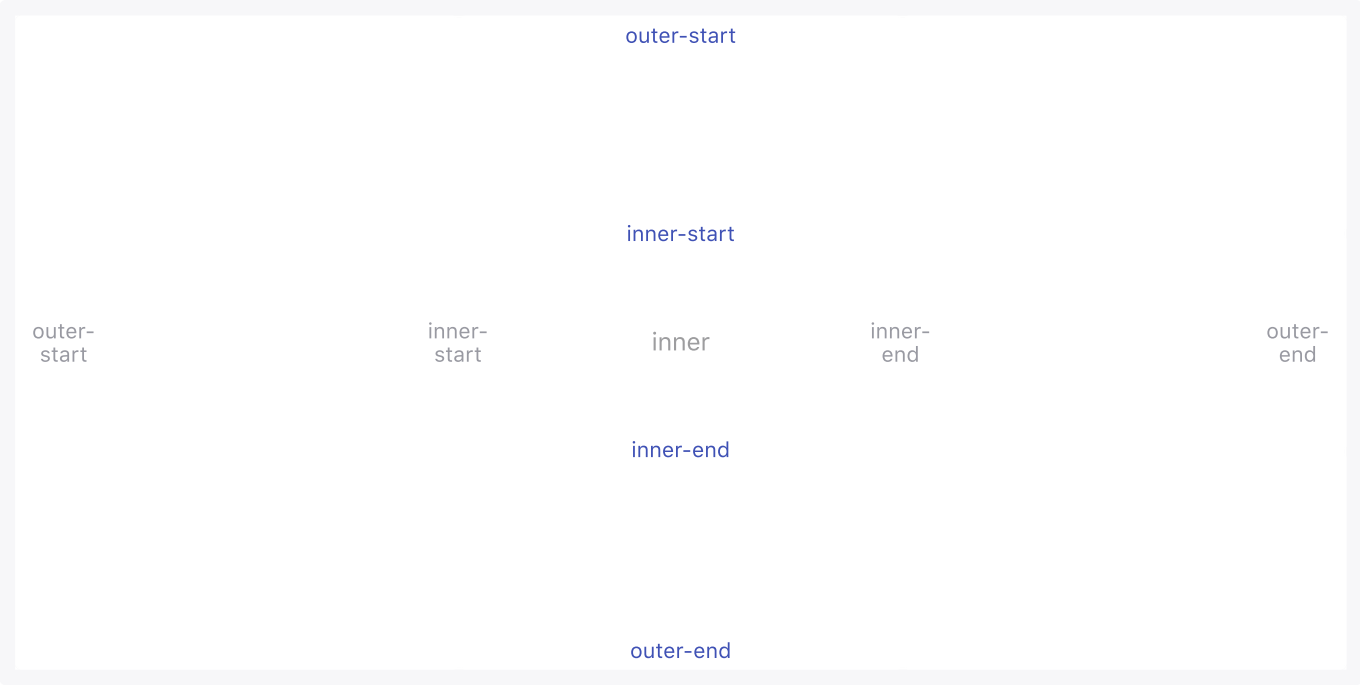
which can be referenced for positioning.

```
grid-template-rows:    [outer-start] 1fr [inner-start] 1fr [inner-end] 1fr [out
grid-template-columns: [outer-start] 1fr [inner-start] 1fr [inner-end] 1fr [out
```

In this example, both rows and columns have `inner-start` and `inner-end` lines, which implicitly assigns the grid area's name as `inner`.

```
grid-area: inner
```

Grid items can then be positioned by the grid area name as opposed to line names.



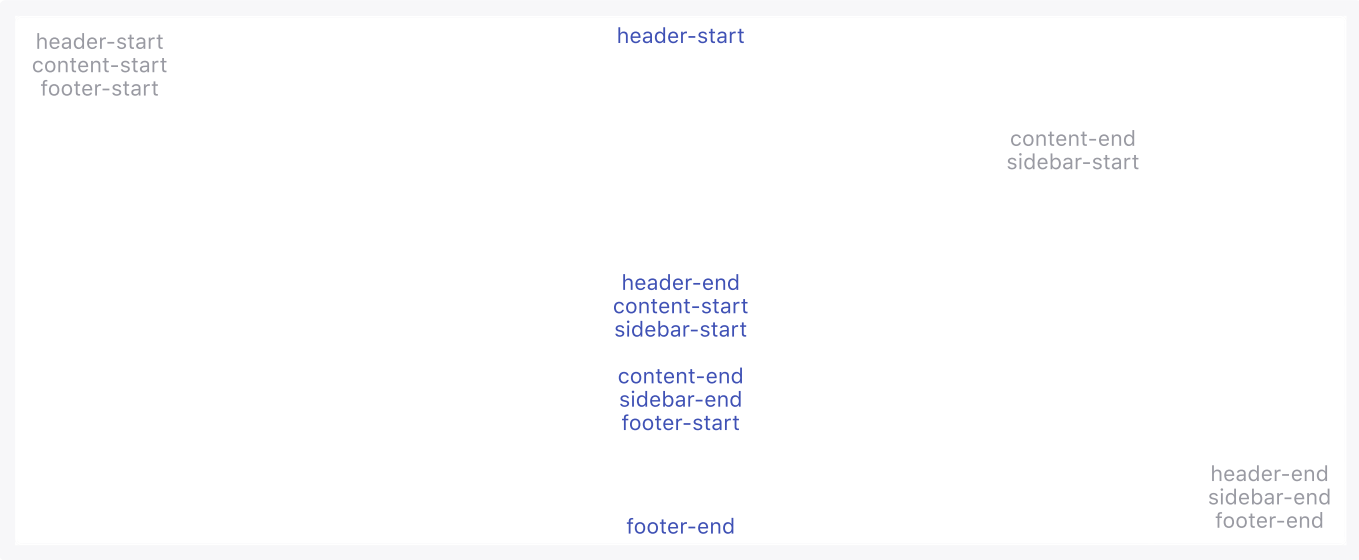
Implicitly Named Grid Lines

Implicitly named grid lines work in reverse to implicitly named grid areas—naming grid areas implicitly assigns names to grid lines.

```
grid-template-areas:   "header header"
                       "content sidebar"
```

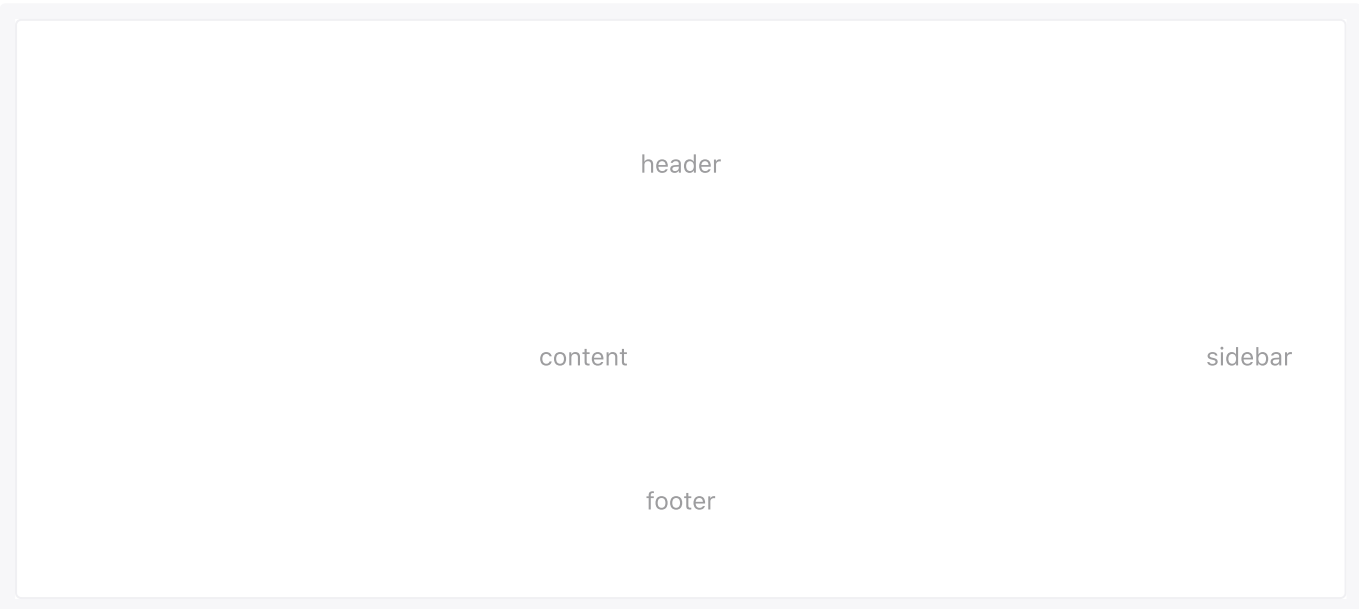
```
                        "footer footer";
grid-template-rows:     80px 1fr 40px;
grid-template-columns: 1fr 200px;
```

Named grid areas will implicitly name the grid lines along the edges of the area. Those grid lines will be named based on the area name and suffixed with `-start` or `-end`.




```
grid-row-start:    header-start;
grid-row-end:      content-start;
grid-column-start: footer-start;
grid-column-end:   sidebar-end;
```

In this example, the header was positioned using the implicit grid line names.



Layering Grid Items

Grid items can be layered/stacked by properly positioning them and assigning `z-index` when necessary.

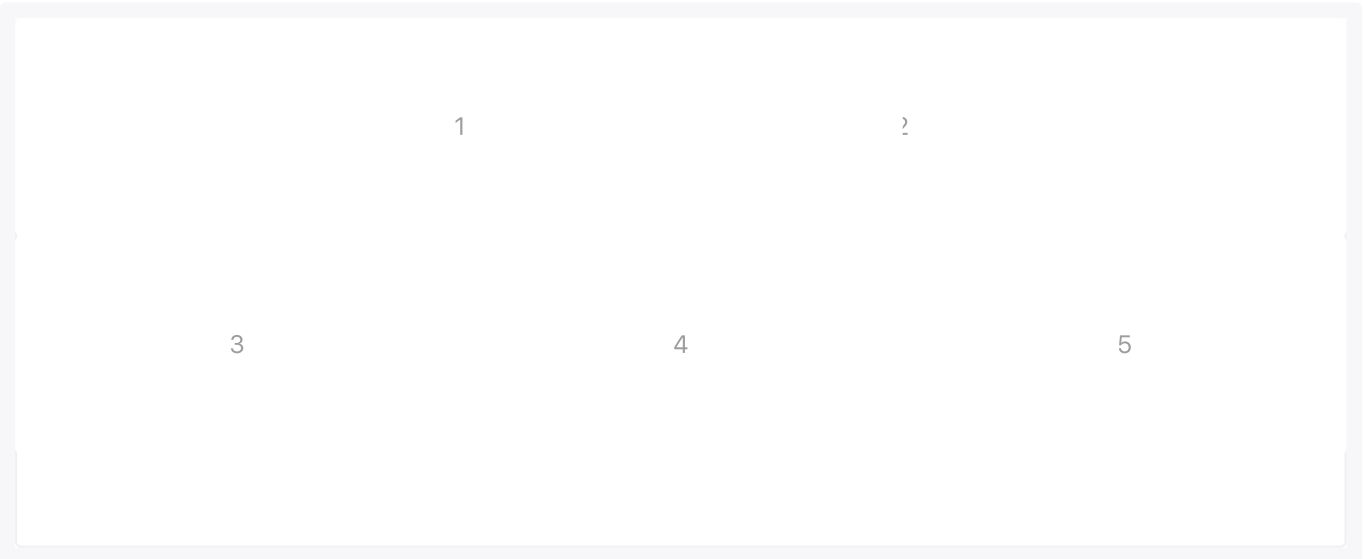
```
.item-1,
.item-2 {
  grid-row-start:  1;
  grid-column-end: span 2;
}

.item-1 { grid-column-start: 1; z-index: 1; }
.item-2 { grid-column-start: 2 }
```

In this example, items 1 and 2 are positioned to start on row line 1 and set to span 2 columns.

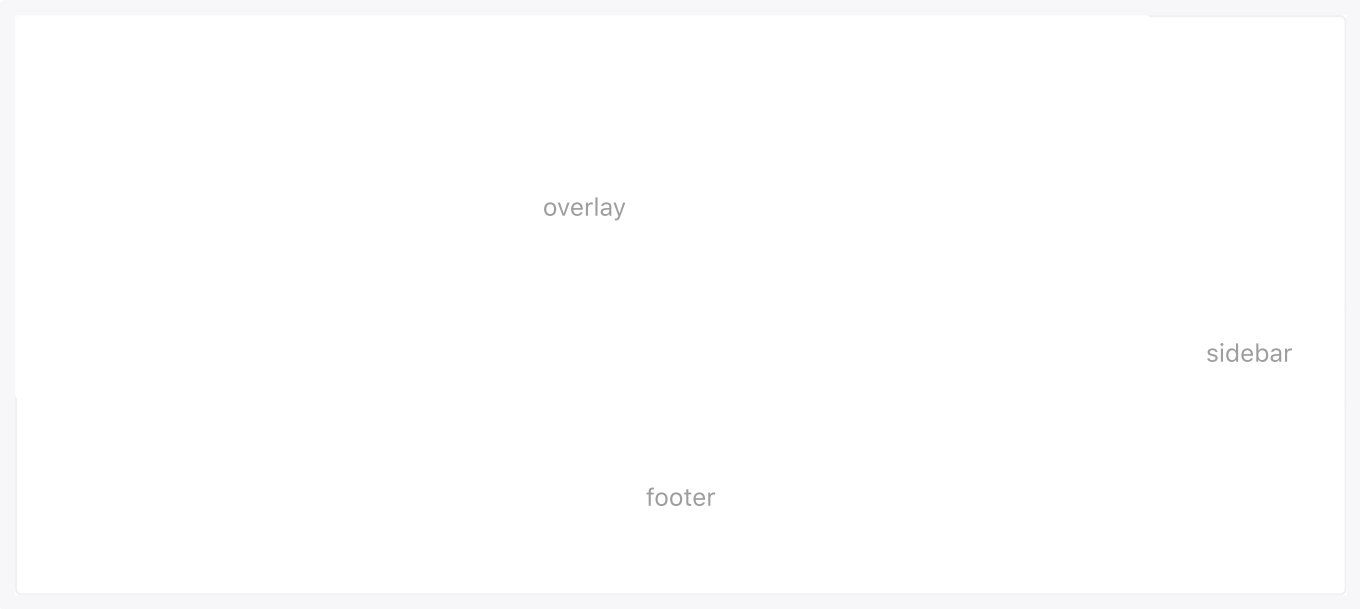
Both items are positioned by grid line numbers. Item 1 is set to start at column line 1, and item 2 at column line 2, which results in both items overlapping in the center column track.

By default, item 2 would sit on top of item 1; however, we've created stacking context by assigning `z-index: 1` to item 1, resulting it to sit on top of item 2.



```
grid-row-start:    header-start;
grid-row-end:      content-end;
grid-column-start: content-start;
grid-column-end:   sidebar-start;
z-index: 1;
```

In this example, a grid item is positioned and layered on top using implicit grid line names from the defined `grid-template-areas`.



Aligning Grid Items (Box Alignment)

CSS's Box Alignment Module complements CSS Grid to allow items to be aligned along the row of column axis.

`justify-items` and `justify-self` align items along the row axis, and `align-items` and `align-self` align items along the column axis.

`justify-items` and `align-items` are applied to the grid container and support the following values:

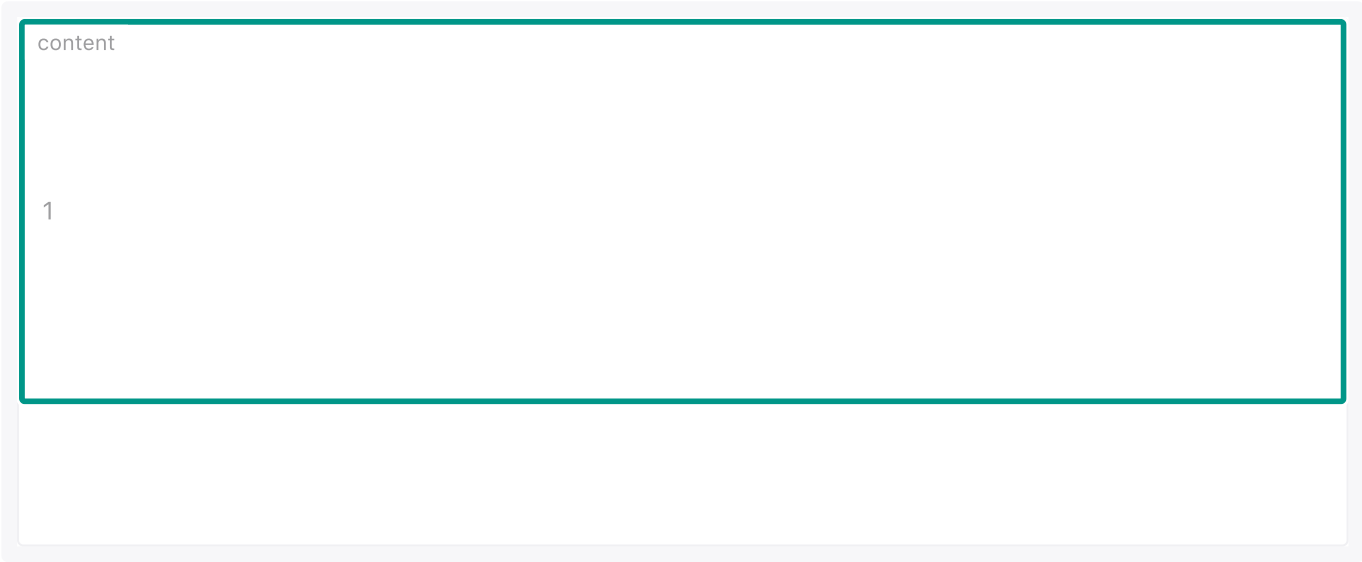
- `auto`
- `normal`
- `start`

- end
- center
- stretch
- baseline
- first baseline
- last baseline

```
.grid {
  grid-template-rows: 80px 80px;
  grid-template-columns: 1fr 1fr;
  grid-template-areas: "content content"
                       "content content";
}
.item { grid-area: content }
```

```
.grid {
  justify-items: start
}
```

Items are positioned at the start of the row axis (row line number 1).




```
justify-items: center
```

Items are positioned at the center of the row axis.

1

`justify-items: end`

Items are positioned at the end of the row axis.

content

1

`justify-items: stretch`

Items are stretched across the entire row axis. `stretch` is the default value.

content

1

`align-items: start`

Items are positioned at the start of the column axis (column line 1).

`align-items: center`

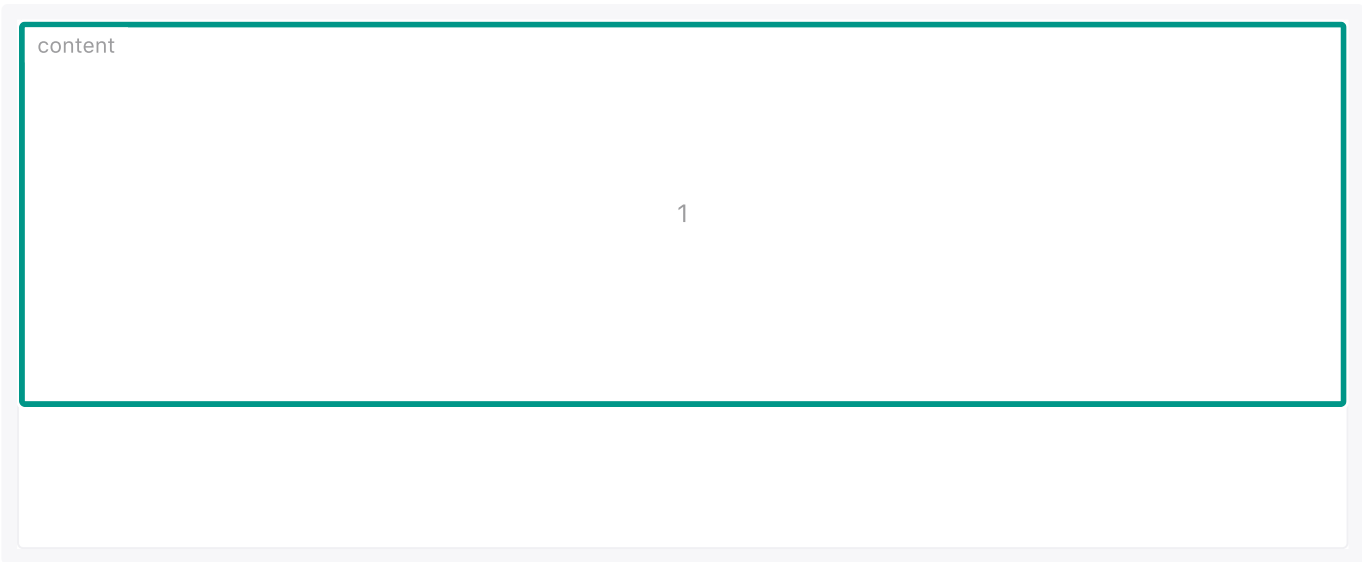
Items are positioned at the center of the column axis.

```
align-items: end
```

Items are positioned at the end of the column axis.

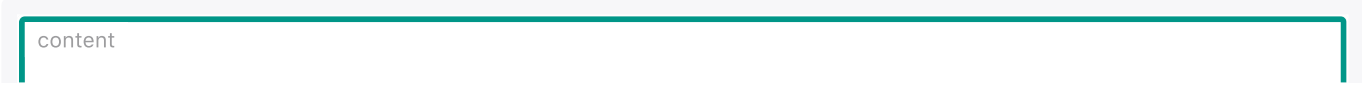

```
align-items: stretch
```

Items are stretched across the entire column axis.




```
justify-items: center
align-items:   center
```

Items are positioned at the center of the row and column axes.



1

Individual items can be self-aligned with the `align-self` and `justify-self` properties. These properties support the following valuse:

- `auto`
- `normal`
- `start`
- `end`
- `center`
- `stretch`
- `baseline`
- `first baseline`
- `last baseline`

```
.item-1 { justify-self: start }
.item-2 { justify-self: center }
.item-3 { justify-self: end }
```

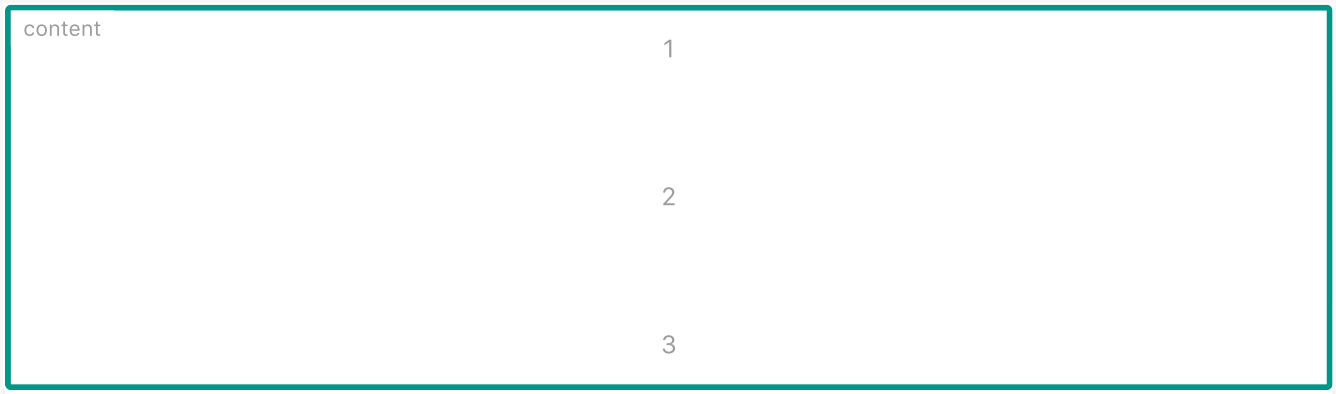
`justify-self` aligns individual items along the row axis.

content

1 2 3

```
.item-1 { align-self: start }
.item-2 { align-self: center }
.item-3 { align-self: end }
```

`align-self` aligns items along the column axis.




```
.item-1 {
  justify-self: center
  align-self:   center
}
```

Item 1 is positioned at the center of the row and column axes.

Aligning Grid Tracks

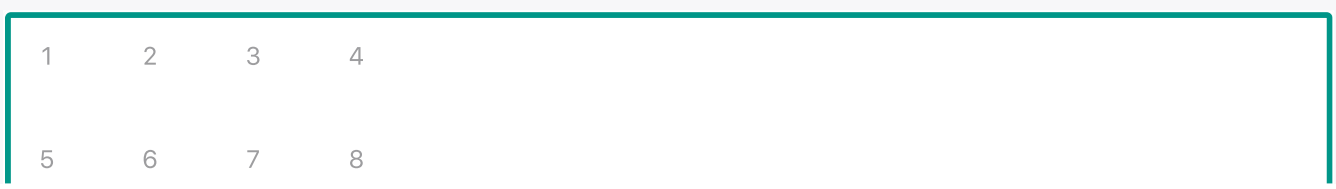
Grid tracks can be aligned relative to the grid container along the row and column axes.

`align-content` aligns tracks along the row axis and `justify-content` along the column axis. They support the following properties:

- `normal`
- `start`
- `end`
- `center`
- `stretch`
- `space-around`
- `space-between`
- `space-evenly`
- `baseline`
- `first baseline`
- `last baseline`

```
.grid {
  width: 100%;
  height: 300px;
  grid-template-columns: repeat(4, 45px);
  grid-template-rows: repeat(4, 45px);
  grid-gap: 0.5em;
  justify-content: start;
}
```

`start` aligns column tracks along and at the start of the row axis—it is the default value.



9 10 11 12

13 14 15 16

```
justify-content: end;
```

Columns are aligned at the end of the row axis.

1 2 3 4

5 6 7 8

9 10 11 12

13 14 15 16

```
justify-content: center;
```

Columns are aligned at the center of the row axis.

1 2 3 4

5 6 7 8

9 10 11 12

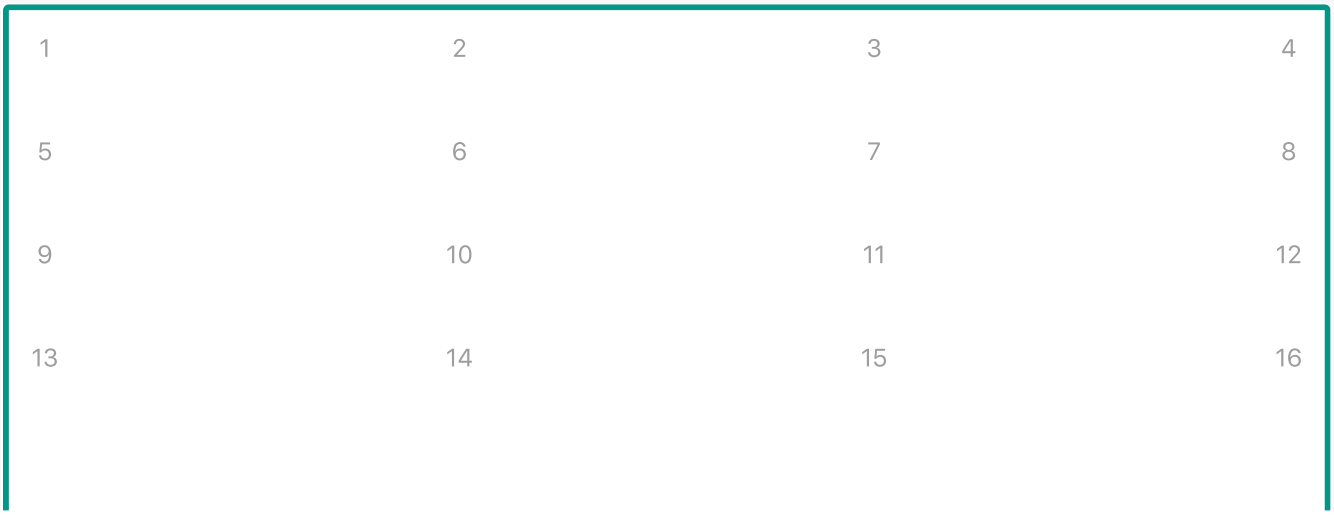
13 14 15 16

```
justify-content: space-around;
```

The remaining space of the grid container is distributed and applied to the start and end of each column track.


```
justify-content: space-between;
```

The remaining space is distributed between the column tracks.



```
justify-content: space-evenly;
```

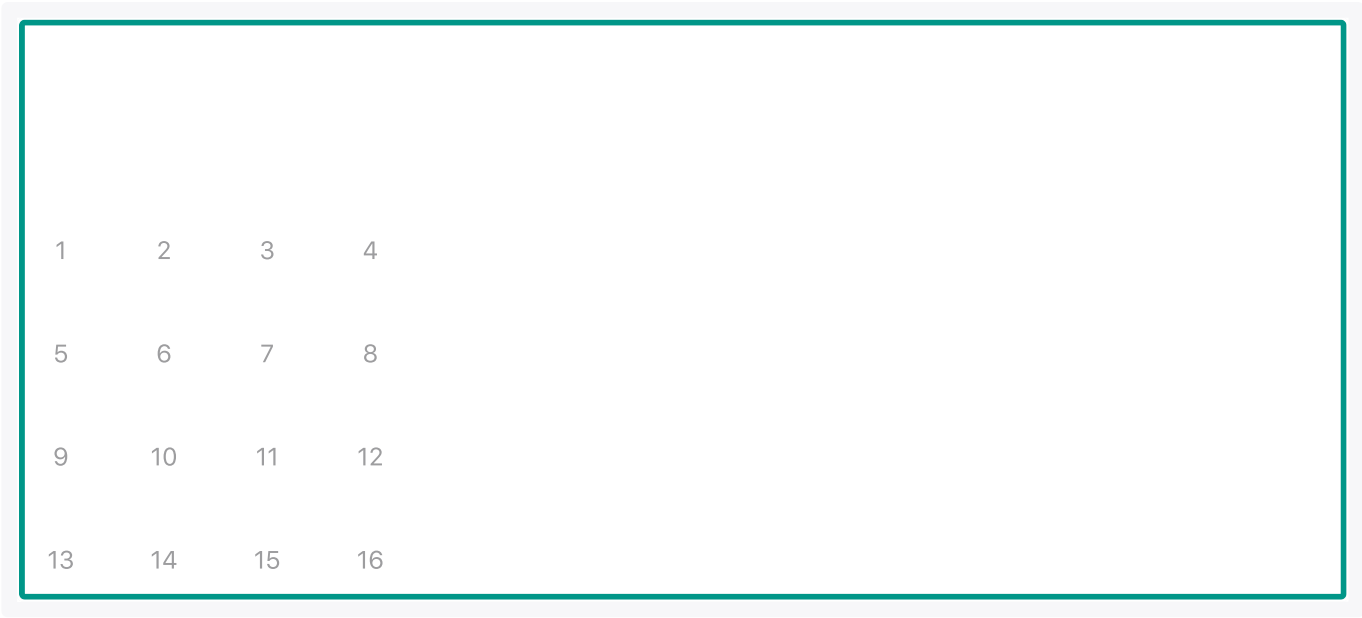
The remaining space is distributed where the space between the columns are equal to the space at the start and end of the row track.


```
align-content: start;
```

`start` aligns rows at the start of the column axis and is the default value.

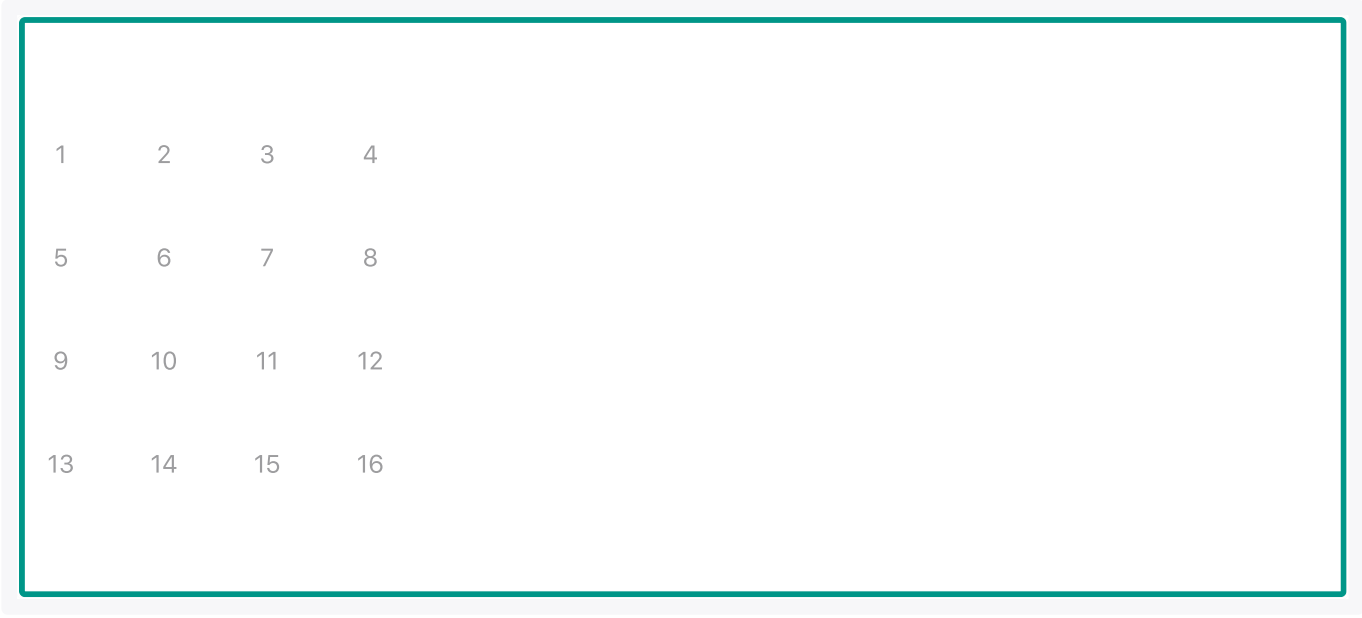
```
align-content: end;
```

Rows are aligned at the end of the column axis.




```
align-content: center;
```

Rows are aligned at the center of the column axis.




```
align-content: space-around;
```

The remaining space of the grid container is distributed and applied to the start and end of each row track.


```
align-content: space-between;
```

The remaining space is distributed between the row tracks.


```
align-content: space-evenly;
```

The remaining space is distributed where the space between the rows are equal to the space at the start and end of the column track.

5 6 7 8

9 10 11 12

13 14 15 16

This guide is designed to give you a fairly comprehensive overview of Grid; however, it doesn't pretend to be a complete technical documentation. Be sure to check out the specs of Mozilla Developer Network and W3C for an even deeper dive.

Here are some other fantastic resources on CSS Grid:

- Complete Guide to Grid on CSS Tricks
- Grid by Example by Rachel Andrew
- The CSS Workshop by Jen Simmons
- Grid Garden by Codepip
- Spring Into CSS Grid by Joni Trythall

I'm susceptible to making mistakes or being wrong. If you see a typo or a mistake, please reach out to me on Twitter or via email.

Huge thank you to Mozilla Developer Network and W3C for the CSS Grid resources; ladies Jen Simmons and Rachel Andrew, who are major contributors to Grid, and it wouldn't be where it's at without them; and my amazing company, Planning Center , for allowing me the time to dive in and learn CSS Grid during Free Week.

Learn CSS Grid by Jonathan Suh · @jonsuh