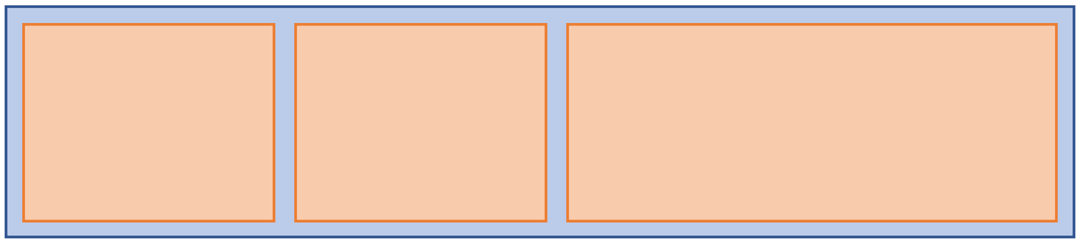
# CSS Grid

CSS grid is a newer way of building a layout using CSS. It allows you to build complex layouts because it takes into account both rows *and* columns. This means that for one grid container, the grid items can be displayed in a complex grid. (Recall that flexbox only allowed you to create a layout in one direction per container—either a row **or** a column.)

Similar to flexbox, for CSS grid you have an outer container (**grid container**—blue box) with the elements you want to set sizes for in your grid as the children of the container (**grid items**—orange boxes). Also similar to flexbox, there are specific properties which apply to the grid container and specific properties which apply to the grid items only. (Note: There is a subgrid functionality in the works but is still *very* new so I wouldn't use it anywhere.)

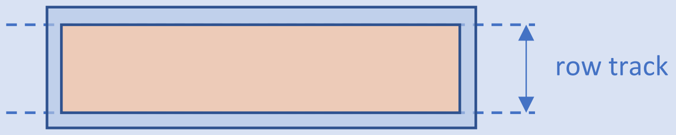


*Track size*

When reading about CSS grid, you'll come across the term "track size" quite often. This refers to the size(s) of your row(s) and/or column(s).

If you consider drawing a horizontal track (i.e. like a road) and draw another track intersecting it perpendicularly, the intersecting part determines a grid box. This is what determines the grid item size if defining both a track size for a row and for a column.

If only defining track sizes for rows, these would determine your row heights. Similarly, if you only define track sizes for columns, these sizes are what determine column widths.



In the diagram above, there is only one row track size defined for the grid container and only one grid item. The grid item takes the row height as defined by the track size for the row.

Let's try a simple example by creating a grid container containing some grid items (see next page).

```
<div class="container">
  <header id="header">Header</header>
  <main id="main">Main</main>
  <aside id="sidebar-one">Sidebar one</aside>
  <aside id="sidebar-two">Sidebar two</aside>
  <footer id="footer">Footer</footer>
</div>
```

Now we can flesh out some of our grid container properties.

## *Grid **container** properties*

***display:*** <*grid*|*inline-grid*>

The display property is **required** to use CSS grid. Possible grid container values are **grid** or **inline-grid.** Inline-grid will shrink the width of the container to the size of the longest content (like how inline-block acts).

For our example, we can use:

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

***grid-template-columns*** *or* ***grid-template-rows***

These properties set the column and row size(s), respectively. If you have multiple columns, you can set multiple lengths separated by spaces.

```
.container {
  display:grid;
  grid-template-columns: 150px 500px 150px; /* set 3 column sizes */
}
```

You can do the same for rows. Note that you can use the keyword **`auto`** to set the row height to whatever the content height is.

```
.container {
  display:grid;
  grid-template-rows: 150px auto 200px; /* set 3 column sizes */
}
```

***fr** unit*

You can use the "`fr`" unit to split the available space in a row or column using **fractions**. For example, if you set a column to have a **grid-template-columns** value of "`1fr 1fr 1fr`" this defines three columns with equal proportion widths (like what we did with flex:1 0 0). If you want the middle column to have twice the width, you can do:

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

*minmax()*

When defining heights for your rows, you may want to make your pixel length a *minimum* height. That is, if the content is short, you want, say, a 50px height but you want to allow the row height to grow if the content is longer. This is more flexible so that you don't have content bleeding outside its container. You can use the minmax() function in grid-template-rows or grid-template-columns to specify a minimum and maximum value.

```
grid-template-rows: 100px auto minmax(50px, auto);
```

In the line above, the first row is set to an unchangeable 100px (because we know the content will not exceed that height), the second row will auto-adjust to fit the content, while the last row is set to a range from 50px to a maximum of whatever the content's height is.

*repeat()*

```
repeat(<num-of-times-to-repeat>, <track-size>)
```

If you have repeat lengths, you can shorten your CSS by using repeat().

For example, instead of

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

you can write

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

### *grid-template-areas*

This property allows you to define different regions or areas in your grid by name. **You still need to define grid-template-rows** (if there are multiple rows) **and grid-template-columns.**

**Important!** You must only use **one area name for one container**. Reusing an area name on a different grid item will not work because the browser won't know which container goes where. Additionally, a grid area **must be rectangular**. It won't work if you try to define an irregular shape.

For our layout example, let's say we want our header row to be 150px tall and our footer to be 100px tall. We'll leave the row with the main content to have an auto-height because we want it to grow with our content. We can do:

```
.container {
  display:grid;
  grid-template-columns:1fr 1fr 1fr 1fr;
  grid-template-rows:150px minmax(400px, auto) 100px;
  grid-template-areas:
    "header header header header"
    "side1 main main side2"
    "footer footer footer footer";
}
```

Although we use the name "header" multiple times, this is still referring to one container which will span four columns.

Each name can then be used in the styles for our **grid items** and are referred to by using the *grid-area* property. For example:

```
#header {
  grid-area: header;
}
```

We can finish off the other grid items for our layout example:

```
#sidebar-one {
  grid-area: side1;
}
#sidebar-two {
  grid-area: side2;
}
#main {
  grid-area: main;
}
#footer {
  grid-area: footer;
}
```

### ***column-gap*** *and* ***row-gap***

You can specify gutter spacing between rows and columns using the row-gap and column-gap properties, respectively.

### ***grid-auto-rows*** *and* ***grid-auto-columns***

Using grid-template-rows and grid-template-columns will create an explicit grid, so any grid item that falls outside of those settings will not follow any track sizes set for those rows and/or columns (they will auto-calculate, so the usual issue would be the row heights). For example, if the template rows and template columns define sizes for cells in a 2x2 grid, if there is a fifth item, the fifth item will auto-calculate row height to follow the content height for that fifth item. To *implicitly* set a row height for all rows in a grid without specifying how many rows you're expecting, use grid-auto-rows.

## Grid ***item*** properties

### ***grid-area***

This is used to refer to the template area names defined in the grid container ***grid-template-areas*** property (see above example).

***order***

You can use this to change the order in which grid items are displayed (just like with flexbox).

***grid-column-start/end and grid-row-start/end***

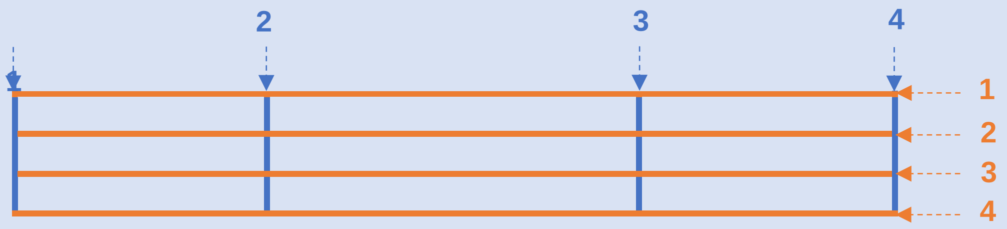
grid-column-start, grid-column-end, grid-row-start and grid-row-end allow you to specify where to place a grid item in the grid based on the grid lines.

Grid line numbers

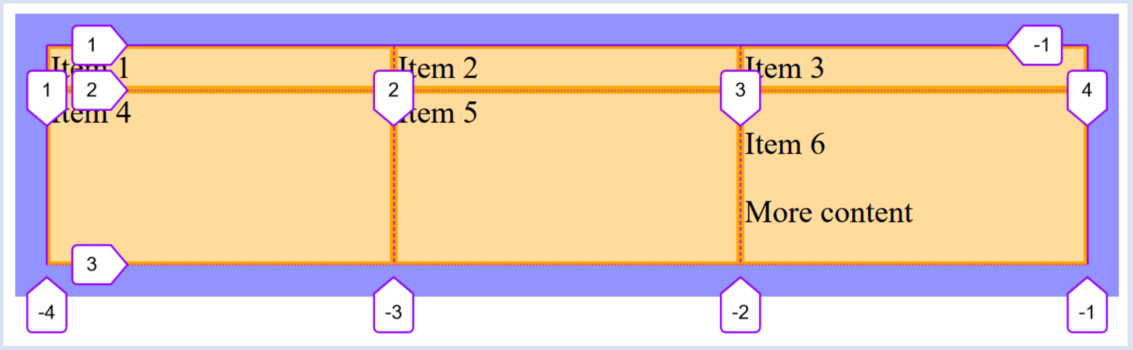
Grid line numbers refer to grid lines in a grid row or column. In the following example of a 3x3 grid, the **row lines have been drawn in orange** and the **column lines have been drawn in blue**.

The row line numbers start from 1 at the top and down to the last row line, so in the diagram below, the final grid row line number is 4.

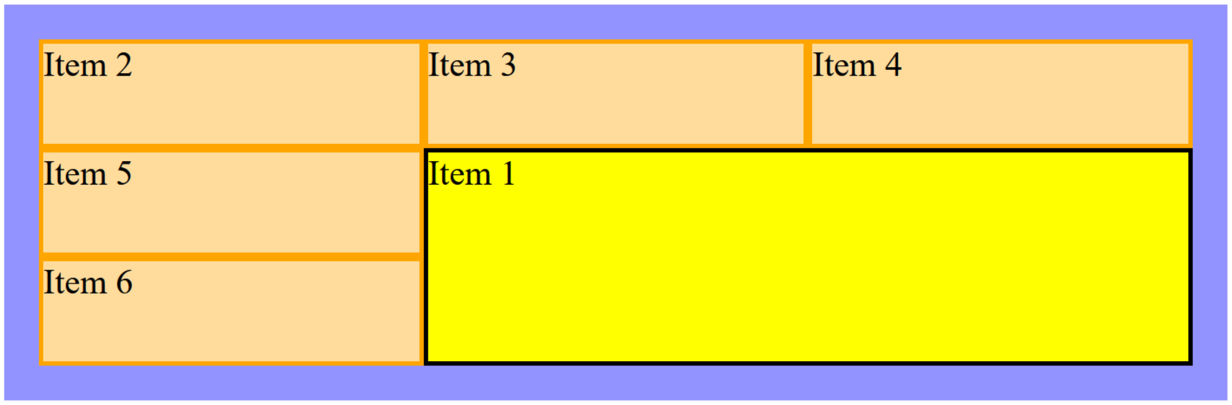
Similarly, the column line numbers start from 1 at the left and count up as you go right.



You can view grid line numbers in your developer tools. In the element inspector, you should see something that looks like a button that says "grid" next to the grid container element. When you click on that you should see the lines.



**Example.** In the following example, .item-1 has been placed in a specific spot in the grid by specific grid line numbers.



```
.container {
  background-color: #9393ff;
  padding:1em;
  display:grid;
  grid-template-columns: repeat(3, 1fr);
  grid-template-rows: repeat(3, 50px);
}
.item {
  background-color: #ffdc9c;
  border:2px solid orange;
}
.item-1 {
  background-color: yellow;
  border:2px solid #000;
  grid-row-start: 2;
  grid-row-end: 4;
  grid-column-start: 2;
  grid-column-end: 4;
}
```

*Grid lines for named template areas*

If you have named template areas the start and end grid lines for those areas are named ***<template-area>*-start** and ***<template-area>*-end**, respectively, for both the row grid lines and column grid lines.

For example, in the header template area in the example on pages 3 to 4, the row start grid line is **header-start** and row end grid line is **header-end**. Similarly, the column start grid line is **header-start** and the column end grid line is **header-end**.

### *grid-column and grid-row*

**grid-column:** *<column-start-line>* **/** *<column-end-line>*;

grid-column is the shortform version of grid-column-start and grid-column-end. In the above example, instead of writing

```
grid-column-start: 2;
grid-column-end: 4;
```

this can be shortened to:

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

**grid-row**: *<row-start-line>* **/** *<row-end-line>*;

For a more complete breakdown of CSS grid-related properties take a look at the following resources:

- https://css-tricks.com/snippets/css/complete-guide-grid/
- https://learncssgrid.com/
- https://css-tricks.com/auto-sizing-columns-css-grid-auto-fill-vs-auto-fit/
- https://cssgridgarden.com/ (a CSS grid game to give you practice with grid properties)

### *If I use grid do I need flexbox and vice versa?*

The answer is: it depends on what you're trying to do. Flexbox and grid are **not** mutually exclusive. Depending on your use case, you may use one or the other or you may even use both in different parts of your page.

Where CSS grid really shines is for complex layouts. For example, in a scenario where you would have two flex containers to control two rows of flex items, you can do this with one grid container.

### *Progressive enhancement and the **@supports** query*

The CSS Grid spec (as it currently is) was completed in November 2017, so it is very new. It is very powerful and allows for very complex layouts. As with anything new, there will not be full support in older browsers so be aware that **you should not ONLY use newer technology** because you may cut out a large chunk of your users.

This does not mean that you can never use newer code, though. You can introduce newer code into your CSS using **progressive enhancement**. This means that you can leave in your older code (and you should for better support), but then as newer techniques are developed you can add in newer code where it would make the most sense to use (i.e. don't just use it for the sake of using it for a simple layout if it is already working as intended). This is possible because CSS is read top-down and if a browser doesn't support a property it just skips over it.

The key is to place the newer properties after the older code **in your CSS rule**. For example, if you are using flexbox, your element would use display:flex. To support grid for newer browsers, in the same rule, you can then add display:grid after the flexbox stuff using a feature query (@supports). Feature queries are only supported for very new browsers, so don't use it for flexbox stuff (it's unnecessary anyway).

For example:

```
#sidebar-one {
  /* do the flexbox stuff here */
  /* also any other styles */
}
@supports (display:grid) {
  /* only newer browsers which recognize the @supports query and
   * the grid layout system will use this
   */

  #sidebar-one {
    /* Add your grid stuff here and undo any flexbox styles which
     * interfere with the grid stuff. You can still leave some stuff to use
     * flexbox and only use this query where it's better to use grid where
     * supported.
     */
  }
}
```

The following are good reads:

- https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout/CSS_Grid_and_Progressive_Enhancement
- https://www.smashingmagazine.com/2017/11/css-grid-supporting-browsers-without-grid/