codeinwp

Home / Blog / CSS Grid Tutorial

The Ultimate CSS Grid Tutorial for Beginners (with Interactive Examples)

by Louis Lazaris / January 1, 2023 / web design & development

For some time, many CSS developers had been holding off on incorporating the CSS Grid Layout specification in real projects. This was due to either volatility of the spec, lack of browser support, or possibly not having the time to invest in learning a new layout technique. We're now at the stage that Grid Layout is safe to use in most projects, so I hope this CSS Grid Layout tutorial will help if you haven't yet fully examined this CSS feature.

Similar to the flexbox tutorial that we did previously, this CSS Grid Layout tutorial will include specific interactive demos showcasing many of the features of the Grid Layout spec.

Table of contents:

#Beginner #developer guide: #CSS Grid Layout tutorial with interactive examples 🏁

CLICK TO TWEET

Why use CSS Grid Layout?

Before getting to specific examples of the different properties and values associated with Grid Layout, let's briefly look at what problems Grid Layout is attempting to address. The spec explains:

> *[Grid Layout] provides a mechanism for authors to divide available space for layout into columns and rows using a set of predictable sizing behaviors.*

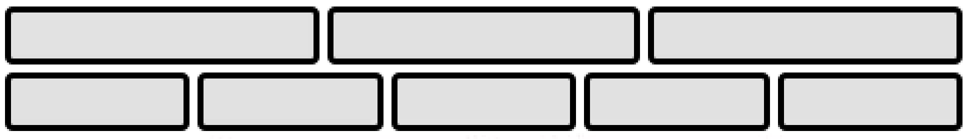


Figure 1 *Representative Flex Layout Example*

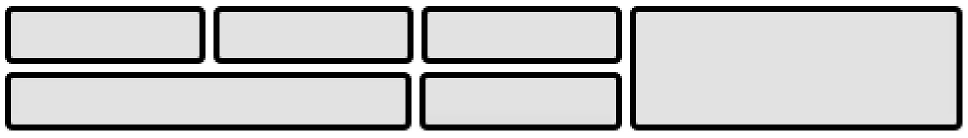


Figure 2 *Representative Grid Layout Example*

designing prototypes or using a CSS grid framework have been accustomed to this sort of thing for years.

Another benefit to Grid is also explained in the spec:

> *Grid Layout allows dramatic transformations in visual layout structure without requiring corresponding markup changes.*

CSS Grid Layout makes it easy to overlap items and has spanning capabilities similar to what some of us did years ago when creating table-based layouts. And best of all, Grid Layout allows you to easily build common designs like a two-column layout with a minimal amount of code.

With those points in mind, let's now get to the meat of this CSS Grid Layout tutorial by looking at the properties and values.

Go to top

CSS Grid Layout terms defined

The spec uses a number of terms to help you grasp the various aspects of Grid Layout and how the grid items function. Here are the key terms you'll want to remember for this CSS Grid tutorial:

- **Grid Container** – The element that establishes the grid and that wraps the grid items
- **Grid Items** – The child elements inside a grid container
- **Grid Lines** – Horizontal and vertical lines that divide the grid
- **Grid Tracks** – Any grid column or row (i.e. what's between grid lines)
- **Grid Cell** – A single cell, like in a table, where a column and row intersects
- **Grid Area** – Any rectangular area of one or more cells, bound by four grid lines

Throughout the following sections I'll use some of the above terms, so keep these in mind as you go through the demos and code.

Go to top

When learning the various parts of the Grid Layout specification, it's useful to remember which features are for the grid container and which are for the grid items.

In this CSS Grid tutorial, I'll first cover the properties you would apply to the grid container. These are as follows:

- `grid-template-rows`
- `grid-template-columns`
- `grid-template-area`

Go to top

Establishing a grid container: `display: grid`

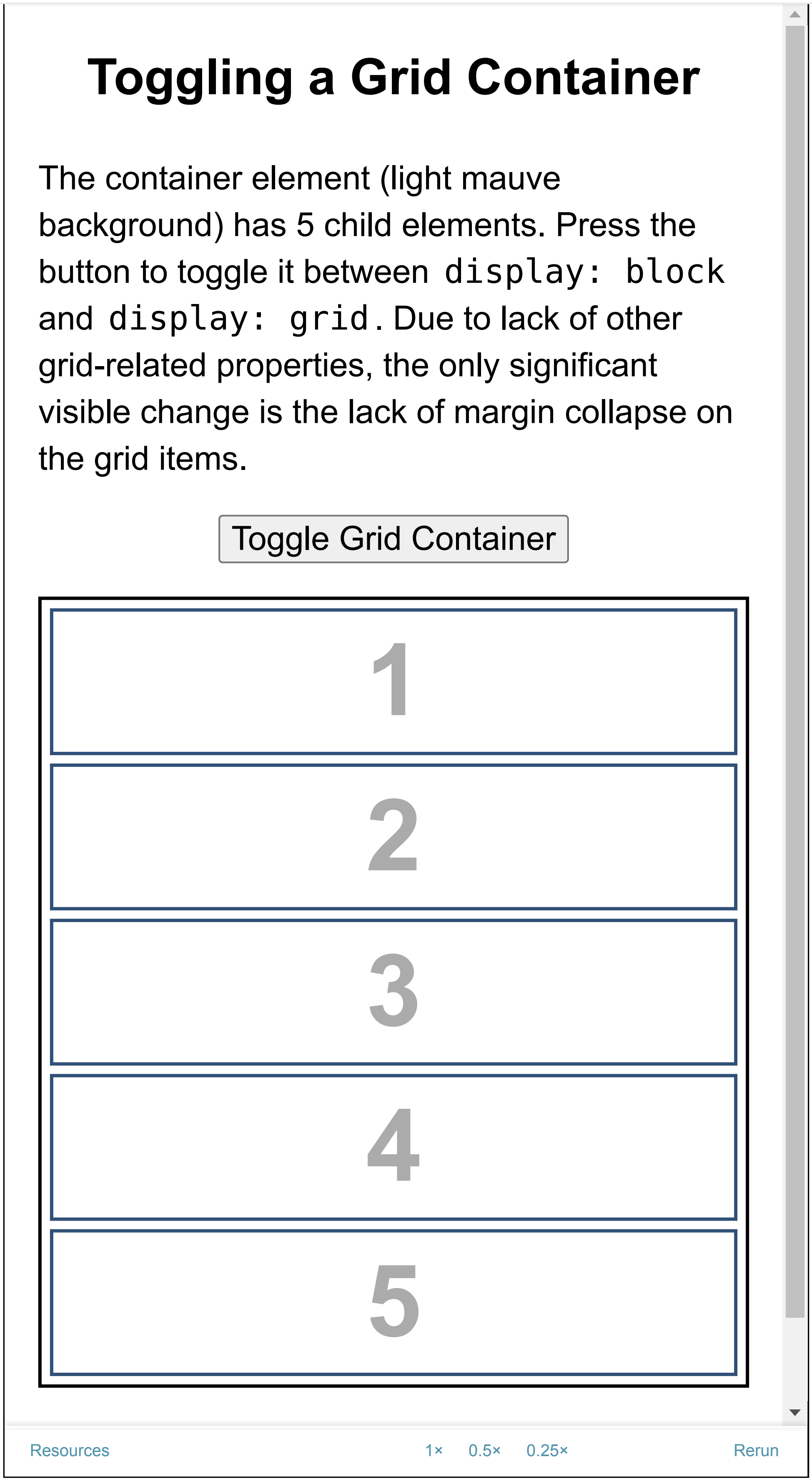
Similar to flexbox, the primary feature that enables Grid Layout capabilities is by means of a specific value for the `display` property.

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

With that code in place, the `.grid-container` element will have the following characteristics:

- Creates a grid formatting context
- Items inside the grid become grid items (similar to the idea of “flex items”)
- Contents comprise a grid, with grid lines forming boundaries around each grid area
- The following properties have no effect on grid items: `float`, `clear`, `vertical-align`
- Margins don't collapse inside a grid container

On its own, `display: grid` won't do a whole lot, which you can see in the following interactive demo:



Toggling a Grid Container

The container element (light mauve background) has 5 child elements. Press the button to toggle it between `display: block` and `display: grid`. Due to lack of other grid-related properties, the only significant visible change is the lack of margin collapse on the grid items.

Notice the only slight change that occurs is that the margins seem to get bigger when you toggle `display: grid`. This is because of the lack of collapsing margins around the grid

Go to top

Defining a grid: `grid-template-rows` and `grid-template-columns`

Once the grid container is in place, you'll want to define the rows and columns for your grid. The `grid-template-rows` and `grid-template-columns` properties define more or less two things:

- The size of the grid tracks (either columns or rows)
- The (optional) line names (discussed in detail later)

A simple example using `px` lengths and no line names would be as follows:

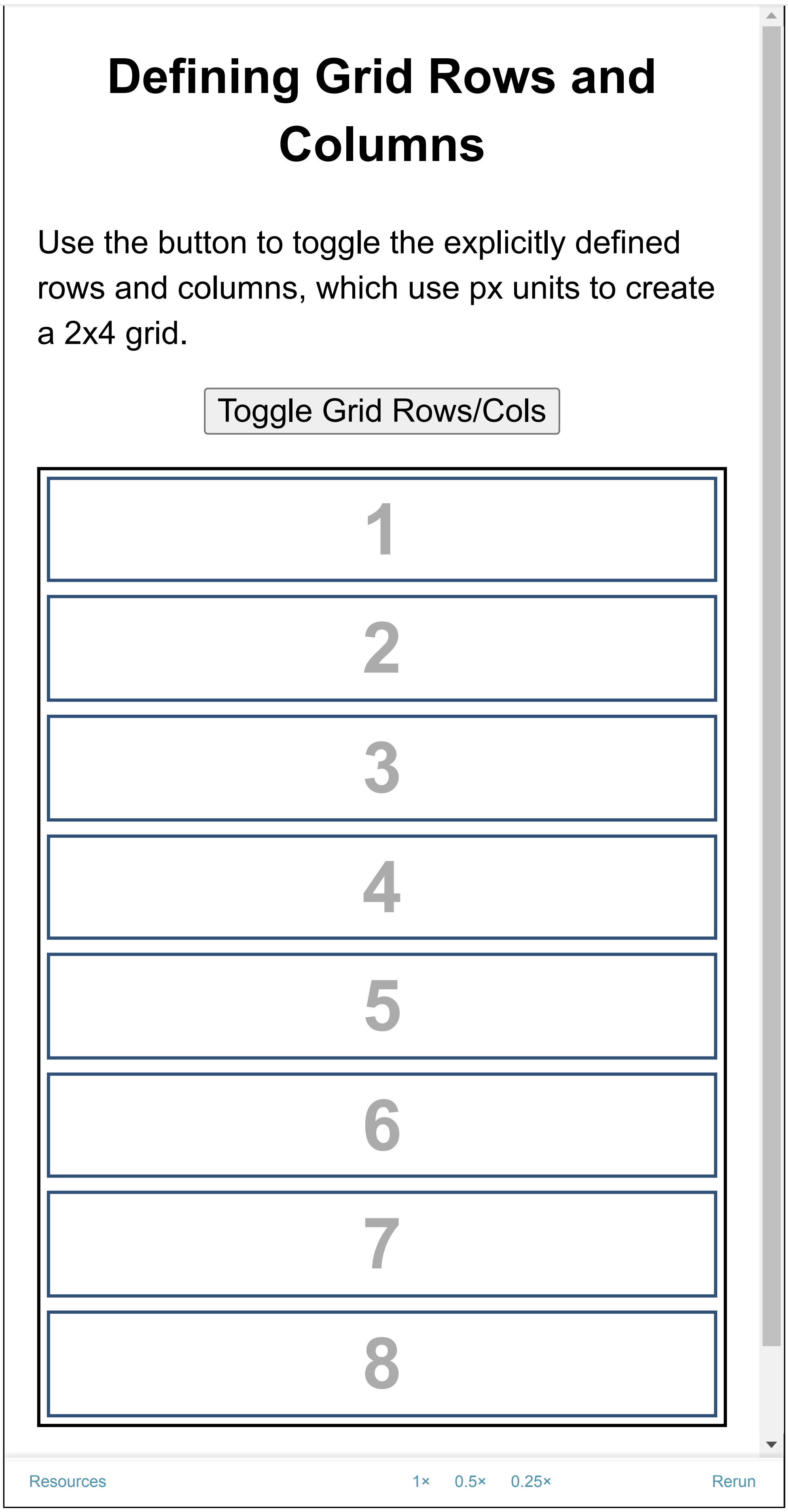
```
.container {
  display: grid;
  grid-template-rows: 120px 120px;
  grid-template-columns: 400px 100px 100px 194px;
}
```

This code tells the browser to render the following:

- two rows each with a height of 120px
- four columns with widths of 400px, 100px, 100px, and 194px, respectively

These are intentionally oddly specific, so this is not ideal. But it serves to demonstrate how the rows and columns are defined. The demo below shows this in action:

codeinwp



Defining Grid Rows and Columns

Use the button to toggle the explicitly defined rows and columns, which use px units to create a 2x4 grid.

Toggle Grid Rows/Cols

1

2

3

4

5

6

7

8

- `fit-content()`
- `minmax(min, max)` – to define a size range for the row or column
- `maxcontent`
- `mincontent`

You may also be interested in:

- Landing Page Basics You Should Know + A Twist to Get Even 1,000% More Opt-Ins
- Average Web Developer Salary: How Much Does a Web Developer Make?
- 10 Ways to Change the Way Your WordPress Site Works With Functions

Go to top

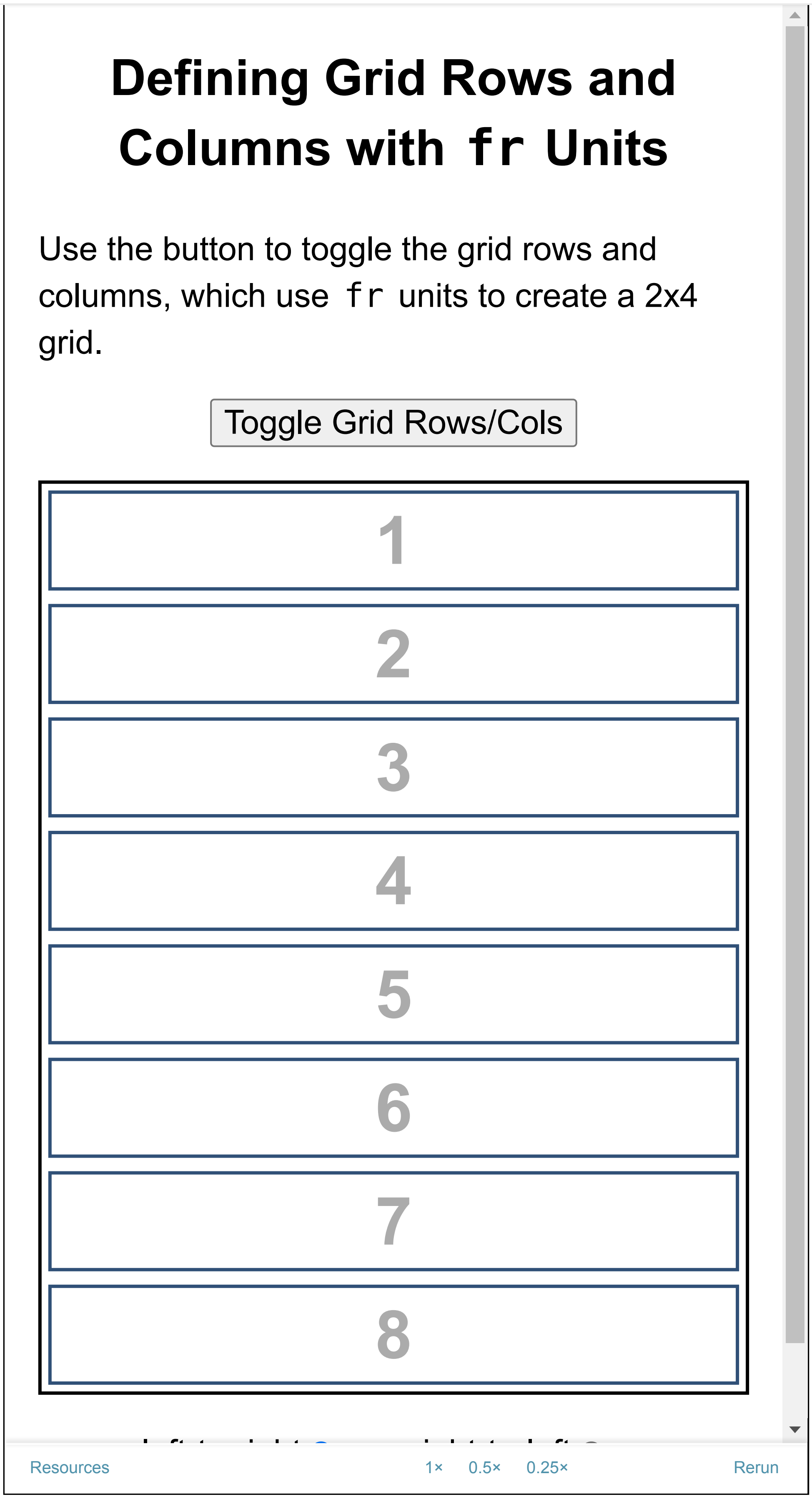
Defining row and column sizes with `fr` units

The `fr` unit, also referred to as a flexible length value, is one of the more useful units you can apply to the `grid-template-rows` and `grid-template-columns` properties.

As the name implies, the `fr` unit represents a fraction of any space left over in a grid container. Here's an example:

```css
.container {
  display: grid;
  grid-template-rows: 1.5fr 1fr;
  grid-template-columns: 3.4fr 1fr 1fr 2fr;
}
```

Similar to the previous examples, this again creates a grid of two rows and four columns. Notice that the `fr` units can be decimal values too. Try it in the demo below:



Defining Grid Rows and Columns with `fr` Units

Use the button to toggle the grid rows and columns, which use `fr` units to create a 2x4 grid.

Keep in mind that the space divided up is based on *available* space, which is calculated after any other layout factors that take priority.

codeinwp

Specifying named grid areas: `grid-template-areas`

Another property that's used on the grid container that you'll find useful is the `grid-template-areas` property. This one takes one or more string values, so it might look a bit strange at first. But it's easy to grasp what it does.

```
.container {
  display: grid;
  grid-template-rows: 1fr 1fr 1fr 1fr;
  grid-template-columns: 1fr 2fr;
  grid-template-areas: "head head"
                       "nav  main"
                       "nav  main"
                       "nav  foot";
}
.item:nth-child(1) {
  grid-area: head;
}
.item:nth-child(2) {
  grid-area: main;
}
.item:nth-child(3) {
  grid-area: nav;
}
.item:nth-child(4) {
  grid-area: foot;
}
```

Each string value in `grid-template-areas` represents a single row in the grid. Each space-separated set of values within each string represents columns in the grid. The names for each grid item in these strings are mapped to specific HTML elements. Notice the above code also uses the `grid-area` property to define which HTML elements will represent each grid area. The next demo should make this a little more clear.

Resources 1× 0.5× 0.25× Rerun

As you can see, this technique requires the following:

- One or more strings representing the rows, columns, and row/column spans
- Grid items with `grid-area` names that match the names defined in the strings

I used the `:nth-child()` pseudo-class to target the elements, but you can use any valid CSS (e.g. classes applied to the elements).

In some cases, you might want one or more grid cells to represent empty space with no content. In that case, you can indicate this with the following syntax (notice the dots):

```
  display: grid;
  grid-template-areas: "head head"
                       ".... main"
                       "nav  ...."
                       "nav  foot";
}
```

You can try it in the following demo:

the space so that the area names line up visually.

Some other things worth noting about using `grid-template-areas`:

- The matching named grid cells in the string need to form a rectangle to be valid (e.g. three cells called “nav” that make an “L” shape would be invalid)
- You have to define all grid cells or else use dots to indicate empty space
- You have to connect like-named areas (e.g. you can’t have “nav” on row one and “nav” on row three)
- The strings in quotes could be on a single line, as long as you separate them by spaces; putting them on different lines helps to visualize the grid

Go to top

Implicitly defined grid tracks: `grid-auto-rows` and `grid-auto-columns`

If a row or column isn’t defined explicitly using the `grid-template-rows` or `grid-template-columns` properties, you have the option to define these *implicitly* using `grid-auto-rows` and `grid-auto-columns`.

The following code demonstrates this:

```
.container {
  display: grid;
  grid-template-areas: "head head"
                       "nav  main"
                       "nav  main"
                       "nav  foot";
  grid-auto-rows: minmax(auto, 75px);
  grid-auto-columns: minmax(auto, 394px);
}
```

In this case, I’m using the `minmax()` function value to define the minimum and maximum values for the rows and columns. With this example, all my rows will be a maximum of 75px and all my columns will be a maximum of 394px, depending on the available space. These values can be any valid length values. You can try the code in the CodePen below:

The grid in that demo starts out using the same grid areas as the previous sections. The toggle button enables the implicit sizing for rows and columns.

In addition to length and percentage values, possible values for `grid-auto-rows` and `grid-auto-columns` include:

- `auto` (the default)
- `min-content`
- `max-content`
- `fit-content()`
- flex values (i.e. using the `fr` unit)
- `minmax(min, max)`

```css
.container {
  grid-auto-rows: max-content auto;
  grid-auto-columns: 2fr 3fr 1fr;
}
```

The above would be the sizes for a grid of two rows and three columns, this being similar to the explicitly sized row and column sizes when defining a grid using `grid-template-rows` and `grid-template-columns`.

Go to top

Auto-arranging grid items: `grid-auto-flow`

One final feature related to auto-sizing that you might find useful is the `grid-auto-flow` property. This property allows you to automatically place grid items into empty spaces, spaces that might be unoccupied because larger items that were prior in source order didn't fit.

Values are:

- `row` – Items are placed to fill each row, adding new rows as needed
- `column` – Items are placed to fill each column, adding new columns as needed
- `dense` – Items are placed to fill in holes as needed, with smaller items that appear later in source order pushed into the empty spots

Try it using the demo below:

Notice how the layout changes based on the value selected, but also take note how the source order changes (the items have numbers shown so you can compare the natural HTML order with the grid order).

Go to top

Grid Layout properties for the grid items

Most of what's been covered so far in this CSS Grid Layout tutorial are the features that you would apply directly to the grid container, which holds the grid items.

The following sections will look at the properties applied to the grid items (i.e. the children of a grid container):

- `grid-row-start`
- `grid-row-end`

- `grid-area`

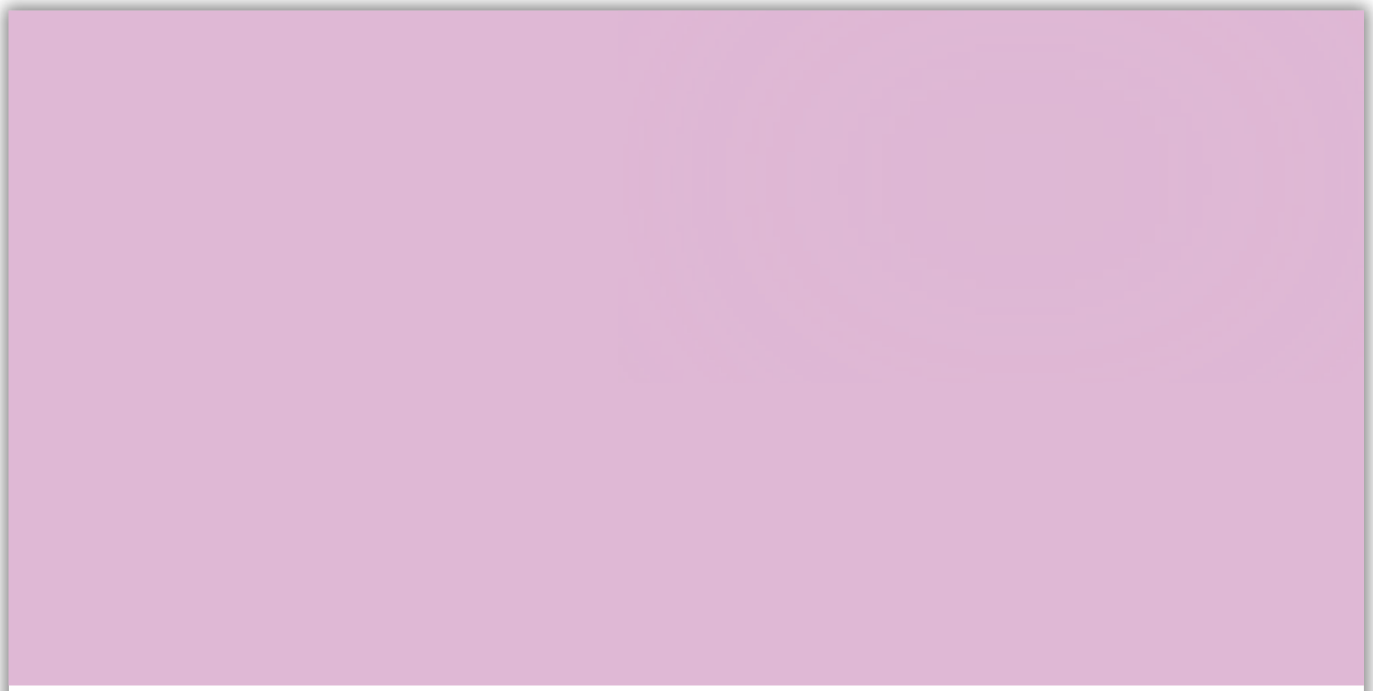
Go to top

Placing Grid items: `grid-row-start` / `end` and `grid-column-start` / `end`

There are four properties that allow you to define where in a grid a specific grid item should start or end:

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

In order to understand how these work, it's important to grasp how the algorithm counts grid lines. Take a look at the following graphic:



Visualizing how the algorithm counts grid lines

The above image displays a 4×4 grid of 16 grid items. The red and blue lines represent the grid lines. The numbers on each line represent how these lines are counted.

You'll notice a few things:

- A single grid cell touches two grid lines

All grid placement properties accept a value of `auto` or one of the following:

- A non-zero positive or negative integer
- A custom identifier that matches a named grid line (more on named grid lines later)
- A combination of the two above values, separated by a space
- The keyword `span` along with an identifier and integer

Here's some example code that I'll use to target two of the grid items (item 1 and item 11):

```
.item:nth-child(1) {
  grid-row-start: 1;
  grid-row-end: 3;
  grid-column-start: 1;
  grid-column-end: 4;
}
.item:nth-child(11) {
  grid-column-start: 2;
  grid-column-end: 4;
  grid-row-start: 4;
  grid-row-end: 6;
}
```

The following interactive demo should help you understand how the above code works. Keep in mind where the grid line numbers start and end (refer back to the graphic) and it should make sense.

When using the `span` keyword with an integer, you can tell a grid item to 'span' over a certain area, relative to its starting point. For example:

```
.item:nth-child(10) {
  grid-row-start: span 3;
```

This tells the 10th grid item to span three full grid rows and two full grid columns from its starting point. This interactive example should help:

Named grid lines

Earlier when I defined a grid I did so with explicit sizing, like this:

```
.container {
  display: grid;
  grid-template-rows: 120px 120px;
  grid-template-columns: 400px 100px 100px 194px;
}
```

With this syntax, you also have the option to name the grid lines so you can reference them when positioning grid items. Put grid line names in square brackets, and you can include multiple names for a single line, separated by spaces inside the brackets (you'll see why this is useful in a moment).

Here's some example code:

```
.container {
  display: grid;
  grid-template-rows: [item1-start] 1fr [item1-end item2-start] 1fr
[item2-end];
  grid-template-columns: [item1-start] 1fr [item1-end item2-start] 1fr
[item2-end item3-start] 1fr [item3-end item4-start] 1fr [item4-end];
}
.item:nth-child(1) {
  grid-row-start: item1-start;  /* same as 1 */
  grid-row-end: item2-end; /* same as 3 */
  grid-column-start: item1-start; /* same as 1 */
  grid-column-end: item3-end; /* same as 4 */
}
.item:nth-child(6) {
  grid-row-start: item2-start; /* same as 2 */
  grid-column-start: item2-start; /* same as 2 */
  grid-column-end: item3-end; /* same as 4 */
}
```

to the custom identifiers that replace the numbers.

I've given many grid lines more than one name. You can see how this can be handy when you're referencing the "start" or "end" of an item. And I've also used the suffixes `*-start` and `*-end` in my naming conventions. This is recommended for better maintainability. The names can be anything you want except the word "span", a reserved keyword.

The interactive demo below shows the above code in action:

Note that the 6th grid item overlaps the others, based on its defined placement. It has a different background color than the others, to ensure you can see its placement clearly. Feel free to fiddle around with the named line values to see how the items change.

Go to top

Common alignment properties

the Box Alignment module.

I won't go into great detail on these, but you can refer back to my flexbox tutorial for interactive demos on many of these features. They work similarly in a Grid Layout context.

- `row-gap` and `column-gap` are applied to the grid container to define gutters between grid rows and columns
- `justify-items` is applied to the grid container to define justification of grid items along the row axis, within the individual grid cells
- `justify-self` is applied to any grid item to define row-axis justification within its individual grid cell
- `align-items` is applied to the grid container to define justification of grid items along the column axis, within the individual grid cells
- `align-self` is applied to any grid item to define column-axis justification within its individual grid cell
- `justify-content` is applied to the grid container to determine how to distribute unused space inside the container along the row axis
- `align-content` is applied to the grid container to determine how to distribute unused space inside the container along the column axis
- `order` is applied to individual grid items to change the order that the items appear by default in the source

Some of these general alignment features are more useful in a flexbox context, so it shouldn't surprise you if you don't use them much in Grid Layout.

Go to top

Grid shorthand properties

Throughout this CSS Grid Layout tutorial, I've used the longhand CSS properties exclusively. This is good when you're learning, and might also be better for code maintenance.

But the Grid Layout specification includes a number of shorthand properties that let you define your grids with a shorter syntax. I'll list all these here along with the longhand properties that they define.

Note that some of these shorthand properties accept keywords along with the represented longhand properties. Some also use a forward slash (/) in between values.

[grid-template-areas]

- The `grid` shorthand is written in one of the following ways (note the keywords allowed):
  - [grid-template]
  - [grid-template-rows] / `auto-flow` [grid-auto-columns]
  - [grid-template-rows] / `auto-flow dense` [grid-auto-columns]
  - `auto-flow` / [grid-auto-rows] / [grid-template-columns]
- `grid-row` – [grid-row-start] / [grid-row-end]
- `grid-column` – [grid-column-start] / [grid-column-end]
- `grid-area` – [grid-row-start] / [grid-column-start] / [grid-row-end] / [grid-column-end]
- `gap` – [row-gap] [column-gap]

Go to top

Other CSS Grid Layout features

There's a lot this tutorial hasn't covered – and rightly so, this is a CSS Grid tutorial for beginners. But there are quite a few features and techniques related to Grid Layout that you'll want to look into once you get the basics down. Here are some links:

- Inline Grids – The `display` property accepts a value of `inline-grid`
- Subgrids – Lets you define a new grid within a single grid area
- Masonry Layout – A popular layout technique used in modern designs, now part of the Grid Layout spec
- Repeating Rows and Columns – Using the `repeat()` function notation, which I only briefly touched on

Go to top

Conclusion

That's it for this deep-dive tutorial into the basics of CSS Grid Layout. I hope the examples in the interactive demos will give you enough to fiddle around with the different properties and values to fully grasp how you can use them to build modern layouts. For convenience, all the CodePen demos from this CSS Grid tutorial can found in this CodePen collection.

By the way, in case you're interested in learning how to craft a website on WordPress, we have a dedicated guide covering this topic as well. Some might even call it the ultimate guide to making a site with WordPress! 😉

Check out more of our web development tutorials:
👉 Webpack tutorial for beginners
👉 CSS Flexbox for beginners
👉 Fetch API tutorial for beginners
👉 Micro-interactions tutorial for beginner developers
👉 Tailwind CSS Tutorial for beginners

Interactive tutorial for #beginner #devs: #CSS Grid Layout (with interactive examples) 🏁🏁🏁

CLICK TO TWEET

...

Don't forget to join our crash course on speeding up your WordPress site. Learn more below:

Layout and presentation by Chris Fitzgerald and Karol K.

Was this article helpful? Yes No

SHOW COMMENTS

Or start the conversation in our Facebook group for WordPress professionals. Find answers, share tips, and get help from other WordPress experts. Join now (it's free)!

10+ Best Themes for Thrive Architect in 2023 (Full Breakdown)

15 Best Elementor Templates for March 2023

10 Best Free Shopify Themes of 2023 – Beautiful & ...

Website Builder Comparison Chart: 7 Best Website Building Tools and ...

Best Website Design Software on the Market (15 Options Analyzed)

10+ Best Laravel Admin Templates for 2023 (Free and Premium)

codeinwp

Squarespace vs WordPress: Which Is Best for Making a Website ...

15 Best Adobe Muse Templates to Kick-Start Your Next Project ...

Wix Tutorial: A Step-by-Step Guide for Beginners on How to ...

FEATURED ON

CodeinWP stands for all-things-WordPress. From web design to freelancing and from development to business, your questions are covered.

- - -

OUR NETWORK

EDITORS’ PICKS

codeinwp