**CSS Grid Tutorial For Beginners –**

[Shahed Nasser](https://shahednasser.medium.com/?source=post_page-----50a94c21b4d8--------------------------------) Jan 5, 2022

CSS Grid Layout is immensely helpful in dividing up elements or the content of a page in rows and columns. It allows you to split up an area between components and define their size, position, placement, and more.

In this article, I’ll go over the basics of CSS Grid for absolute beginners who are looking to understand it and grasp the concept to use it in their projects.

**Why Use CSS Grid**

CSS Grids allow you to place items in a container while maintaining their layout and positions. With the Grid Layout, you have more control over where should each item be, what its width is, and how it should be placed in a container.

Think of a webpage and how there’s usually a navigation bar, a sidebar, and the content in the middle. A CSS Grid can help you achieve that sort of styling easily by dividing up the page into different sections where each element can take place in the specified section.

CSS Grids can also be used in elements in a webpage rather than the entire webpage. Let’s say you have a gallery and you want the images in the gallery to be all of the same sizes. You also want a specific number of images per row. You can control all of that using a CSS Grid.

A lot of beginners find the concept of CSS Grid Layout to be confusing. This article will hopefully help you understand the different properties that come with CSS Grid and how you can use them.

**Your First Grid**

Let’s say we have the following HTML:

<div *class*="container">  
 <p *class*="red"></p>  
 <p *class*="green"></p>  
 <p *class*="blue"></p>  
</div>

We want to make the div with the class container a grid where we can control the elements inside as explained earlier.

All we need to do is change the display property of the class container:

.container {  
 display: grid;  
}

This will make the element with the class container a grid element. If you test this out, you'll see that you can see nothing. The reason behind that is the p elements in the .container grid don't have any elements in them or width or height.

Add the following CSS:

.container p {  
 width: 100px;  
 height: 100px;  
}  
  
.red {  
 background-color: red;  
}  
  
.green {  
 background-color: green;  
}  
  
.blue {  
 background-color: blue;  
}

This will set the width and height of the p elements to 100px. It will also give a different background color for each element.

If you check your grid now it should look something like this:

Shape

Description automatically generated

This is your first grid! As you can see, there’s nothing fancy here as we haven’t yet used any grid properties to place the elements or control their size.

Instead of specifying the width and height of each element, you can also use grid-auto-rows and grid-auto-columns on the grid element to specify the default height and width of elements:

.container {  
 ...  
 grid-auto-rows: 100px;  
 grid-auto-columns: 100px;  
}

**Elements Placements**

Grids consist of rows and columns. You can have as many rows or columns as you want in a Grid, and you can assign your elements to certain placements in a grid.

**Columns**

Let’s say you want to place these 3 elements as just 3 items in a row. we can do that using the grid-template-columns property.

This property takes an n number of values, where n is the number of columns you want in the grid and each value is the width of each column.

For example, if you want to have 3 columns in the grid each having its width as 100px you can use:

grid-template-columns: 100px 100px 100px;

This will divide the grid into 3 columns, and each column will be 100px.

Let’s say you want the second column to be 200px:

grid-template-columns: 100px 200px 100px;

If you also don’t have a specific value to apply you can use the value auto:

grid-template-columns: auto auto auto;

Alternatively, instead of repeating the values, you can use the [repeat()](https://developer.mozilla.org/en-US/docs/Web/CSS/repeat()) function:

grid-template-columns: repeat(3, auto);

The repeat CSS function takes the first parameter the number of times an expression should be repeated, and the second parameter the statement to repeat.

Let’s go back to our example, we’ll place our elements in 3 columns, each having an automatic width.

Change the properties of .container to the following:

.container {  
 display: grid;  
 grid-template-columns: repeat(3, auto);  
}

This will ensure the grid has 3 c0lumns with each having their width set to auto.

Next, remove the width property set on the p elements from earlier. It should only have height now:

.container p {  
 height: 100px;  
}

If you check the grid now, it should have 3 elements with equal widths next to each other.

Shape, rectangle

Description automatically generated

**Gap Between Columns**

In a lot of cases, you might not want columns to be right next to each other. To add a gap between columns, you can use the property grid-column-gap which takes the length of the gap between the columns as a value.

Let’s say we want to add a gap of 5px between our columns. Add the following property to .container:

grid-column-gap: 5px;

This will add a 5px space between the grid columns.

Shape, rectangle

Description automatically generated

**Placement in Columns**

You can control the placement of items relative to columns using the grid-column-start and grid-column-end properties. These properties allow you to specify at which column an item in a grid should start, and/or at which column an item in a grid should end.

The value for grid-column-start and grid-column-end can be a number. In this case, it will specify the column to start at and the column to end before.

For example, if you want to specify that the .red element should start in the beginning and end before the 3rd column you use the following properties:

.red {  
 background-color: red;  
 grid-column-start: 1;  
 grid-column-end: 3;  
}

Now, the .red element will span 2 columns and the .blue element will be pushed to the row below.

Shape, rectangle

Description automatically generated

Alternatively, you can provide a value to grid-column-start and grid-column-end in the format span n, where n is the number of columns to span. This does not specify where exactly the column should start or end. Instead, it specifies that in its natural place it should take up n columns.

If we want to apply the same example, we can use the following properties for .red instead:

.red {  
 background-color: red;  
 grid-column-start: span 2;  
}

This means the element should span 2 columns. This will result in the same layout as shown previously.

These properties can be used to control the element placements even more. For example, now that the blue element is on its own row let’s place it at the end of it:

.blue {  
 background-color: blue;  
 grid-column-start: 3;  
}

Now, instead of the blue element just being shifted down outside of our control, we specify exactly where it should be:

A picture containing shape

Description automatically generated

**Rows**

You just learned how to place and size our elements in the grid’s columns. Next, you’ll learn how to do the same, but in rows.

To specify how many rows a grid has, you can use grid-template-rows. Similar to columns, the grid-template-rows takes n values, where n is the number of rows the grid will have and the value is the height of each row.

So, if you want to have 2 rows of 100px height you can use the following:

grid-template-rows: 100px 100px;

Alternatively, you can use the repeat function:

grid-template-rows: repeat(2, 100px);

This will create 2 rows each having 100px height.

Let’s go back to our example. In this section, we want to have 2 columns and 2 rows. Change .container to the following:

.container {  
 display: grid;  
 grid-template-columns: repeat(2, auto);  
 grid-template-rows: repeat(2, 100px);  
 grid-column-gap: 5px;  
}

This will create 2 columns of auto width and 2 rows of 100px height.

Next, reset the rest of the styles that we performed earlier:

.red {  
 background-color: red;  
}  
  
.green {  
 background-color: green;  
}  
  
.blue {  
 background-color: blue;  
}

This will just set the background color of each of the child elements. Notice that you don’t need to set the height and width anymore of the elements as we’re applying it using grid-template-columns and grid-template-rows.

If you check your grid now you should have 2 items in one row and an item on the second.

Shape, square

Description automatically generated

**Gap Between Rows**

In a lot of cases, you won’t need the items in different rows right next to each other. You can add space between rows using grid-row-gap.

Let’s say we want to add a 5px gap between rows. Add the following property to the .container class:

grid-row-gap: 5px;

This will add a 5px space between the rows in our grid.

Shape, square

Description automatically generated

**Placement in Rows**

You can use the grid layout to control the placement of elements in rows. To do that, you can use the properties grid-row-start and grid-row-end.

The value of grid-row-start and grid-row-end can be a number. The value of grid-row-start will be the row the element should start on, and the value of grid-row-end will be the row the element should end before.

For example, to make sure that the red element spans 2 rows we can use the following properties:

.red {  
 background-color: red;  
 grid-row-start: 1;  
 grid-row-end: 3;  
}

Chart, shape, treemap chart, square

Description automatically generated

Alternatively, you can provide a value to the 2 properties in the format span n where n is the number of rows the element should span.

So, instead of the previous properties used, you can use the following to make sure the red element spans 2 rows:

.red {  
 background-color: red;  
 grid-row-start: span 2;  
}

This will result in the same layout.

These properties are not only helpful for changing how many rows an element spans. They can also be used to completely control where each element is placed.

Let’s say you want to place the blue column in the first row. You can apply the following styling to do that:

.blue {  
 background-color: blue;  
 grid-row-start: 1;  
}

This will result in the blue element being in the first row. However, as the placement of elements has changed the red element will be on the right.

Shape, treemap chart, square

Description automatically generated

To ensure that the blue item is on the right and the red element remains on the left, you can use the grid-column-start property:

.blue {  
 background-color: blue;  
 grid-row-start: 1;  
 grid-column-start: 2;  
}

This will retain the original placement of the red element, but the blue item will be in the first column.

Chart, shape, treemap chart

Description automatically generated

As you can see, you can use both properties for rows and columns to control many aspects of positioning and sizing of a grid.

**Grid Areas**

Instead of controlling a grid as columns and rows, you can control a grid as areas.

Let’s say we want to implement a header, content, and sidebar layout. The green element will be the header, the red will be the content, and the blue will be the sidebar.

First, start by giving the different element area names using the grid-area property:

.red {  
 background-color: red;  
 grid-area: Content;  
}  
  
.green {  
 background-color: green;  
 grid-area: Header;  
}  
  
.blue {  
 background-color: blue;  
 grid-area: Sidebar;  
}

This gives the .red element the grid-area name Content, the .green element the grid-area name Header, and the .blue element the grid-area Sidebar.

Next, you use the grid-template-areas property on the grid container element to determine how these different areas should be distributed. The grid-template-areas takes as a value n string values, where n is the number of rows in the grid layout you're creating. The value will be the name of the areas separated by space. Each name specifies a column.

So, if you want to have 2 columns in your grid and you want the Header element to take 2 columns in a row, you can use the value "Header Header" for the row.

In our example, we’ll create 2 rows and 3 columns. The first row will just be the Header which will span the 3 columns, and the second row will be the Content which will span 2 columns and the Sidebar which will be just one column:

.container {  
 display: grid;  
 grid-template-areas: "Header Header Header"   
 "Content Content Sidebar";  
 grid-row-gap: 5px;  
 grid-column-gap: 5px;  
 grid-auto-rows: 100px;  
}

Since we want three columns in the grid, each string value in grid-template-areas should have 3 area names. The first string value is for the first row in the grid. As we just want the Header to occupy it the value will be the Header repeated 3 times.

The second string value is for the second row. We want the Content area to span 2 columns. So, in the string value we repeat the Content area twice then the Sidebar.

Notice that each string value is wrapped in " and string values are separated by a space.

We also add a gap between rows and columns using grid-row-gap and grid-column-gap. We also set the default row height to 100px using grid-auto-rows.

This will result in the following layout:

Shape, arrow, rectangle

Description automatically generated

**fr Unit**

To ensure that columns or rows in a grid use the same height or width, you can use the [fr unit](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout/Basic_Concepts_of_Grid_Layout" \l "fixed_and_flexible_track_sizes" \t "_blank). For example, to create 3 columns each having the same width:

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

**Conclusion**

In this tutorial, you learned how CSS Grid Layout works in terms of rows, columns, and areas. Using a Grid Layout in CSS will help you control the size and placement of elements in your page or a container element.