Basic Concepts of grid layout

CSS Grid Layout introduces a two-dimensional grid system to CSS. Grids can be used to lay out major page areas or small user interface elements. This article introduces the CSS Grid Layout and the new terminology that is part of the CSS Grid Layout Level 1 specification. The features shown in this overview will then be explained in greater detail in the rest of this guide.

What is a grid?

A grid is an intersecting set of horizontal and vertical lines – one set defining columns, and the other, rows. Elements can be placed onto the grid, within these column and row lines. CSS grid layout has the following features:

Fixed and flexible track sizes

You can create a grid with fixed track sizes – using pixels for example. This sets the grid to the specified pixel which fits to the layout you desire. You can also create a grid using flexible sizes with percentages or with the new `fr` unit designed for this purpose.

Item placement

You can place items into a precise location on the grid using line numbers, names or by targeting an area of the grid. Grid also contains an algorithm to control the placement of items not given an explicit position on the grid.

Creation of additional tracks to hold content

You can define an explicit grid with grid layout. The Grid Layout specification is flexible enough to add additional rows and columns when needed. Features such as adding “as many columns that will fit into a container” are included.

Alignment control

Grid contains alignment features so we can control how the items align once placed into a grid area, and how the entire grid is aligned.

Control of overlapping content

More than one item can be placed into a grid cell or area and, they can partially overlap each other. This layering may then be controlled with the `z-index` property.

Grid is a powerful specification that, when combined with other parts of CSS such as flexbox, can help you create layouts that were previously impossible to build in CSS. It all starts by creating a grid in your **grid container**.

The Grid container

We create a *grid container* by declaring `display: grid` or `display: inline-grid` on an element. As soon as we do this, all *direct children* of that element become *grid items*.

In this example, I have a containing div with a class of wrapper and, inside are five child elements.

```
<div class="wrapper">
  <div>One</div>
  <div>Two</div>
  <div>Three</div>
  <div>Four</div>
  <div>Five</div>
</div>
```

I make the `.wrapper` a grid container.

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

One

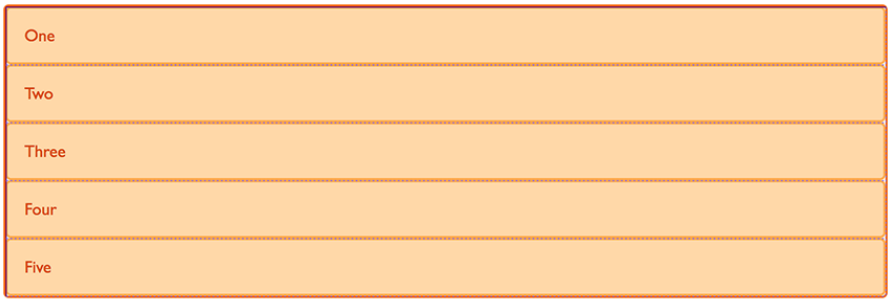
Two

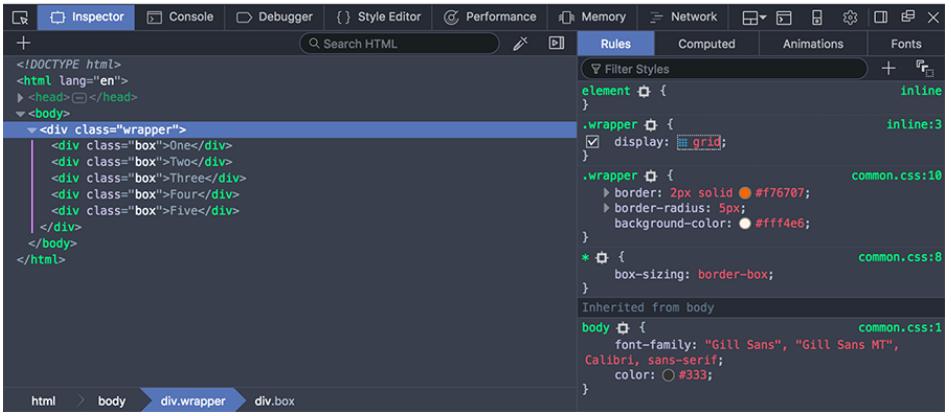
Three

Four

Five

All the direct children are now grid items. In a web browser, you won't see any difference to how these items are displayed before turning them into a grid, as grid has created a single column grid for the items. At this point, you may find it useful to work with the Grid Inspector, available as part of Firefox's Developer Tools. If you view this example in Firefox and inspect the grid, you will see a small icon next to the value `grid`. Click this and then the grid on this element will be overlaid in the browser window.



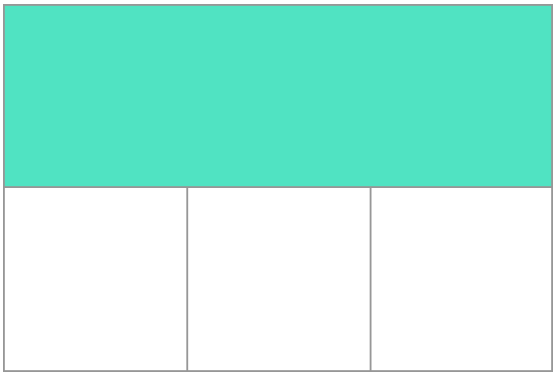


As you learn and then work with the CSS Grid Layout this tool will give you a better idea of what is happening with your grids visually.

If we want to start making this more grid-like we need to add column tracks.

Grid Tracks

We define rows and columns on our grid with the `grid-template-columns` and `grid-template-rows` properties. These define grid tracks. A *grid track* is the space between any two lines on the grid. In the below image you can see a track highlighted – this is the first row track in our grid.



I can add to our earlier example by adding the `grid-template-columns` property, then defining the size of the column tracks.

I have now created a grid with three 200-pixel-wide column tracks. The child items will be laid out on this grid one in each grid cell.

```
<div class="wrapper">
  <div>One</div>
  <div>Two</div>
  <div>Three</div>
  <div>Four</div>
  <div>Five</div>
</div>
```

```
.wrapper {
  display: grid;
  grid-template-columns: 200px 200px 200px;
}
```

The fr Unit

Tracks can be defined using any length unit. Grid also introduces an additional length unit to help us create flexible grid tracks. The new `fr` unit represents a fraction of the available space in the grid container. The next grid definition would create three equal width tracks that grow and shrink according to the available space.

```
<div class="wrapper">
  <div>One</div>
  <div>Two</div>
  <div>Three</div>
  <div>Four</div>
  <div>Five</div>
</div>
```

```
.wrapper {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
}
```

In this next example, we create a definition with a `2fr` track then two `1fr` tracks. The available space is split into four. Two parts are given to the first track and one part each to the next two tracks.

```
.wrapper {
  display: grid;
  grid-template-columns: 2fr 1fr 1fr;
}
```

In this final example, we mix absolute sized tracks with fraction units. The first track is 500 pixels, so the fixed width is taken away from the available space. The remaining space is divided into three and assigned in proportion to the two flexible tracks.

```
.wrapper {
  display: grid;
  grid-template-columns: 500px 1fr 2fr;
}
```

Track listings with `repeat()` notation

Large grids with many tracks can use the `repeat()` notation, to repeat all or a section of the track listing. For example the grid definition:

```
.wrapper {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
}
```

Can also be written as:

```
.wrapper {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
}
```

Repeat notation can be used for a part of the track listing. In this next example I have created a grid with an initial 20-pixel track, then a repeating section of 6 `1fr` tracks then a final 20-pixel track.

```
.wrapper {
  display: grid;
  grid-template-columns: 20px repeat(6, 1fr) 20px;
}
```

Repeat notation takes the track listing, and uses it to create a repeating pattern of tracks. In this next example, my grid will consist of 10 tracks, a `1fr` track, and then followed by a `2fr` track. This pattern will be repeated five times.

```
.wrapper {
  display: grid;

```

```
  grid-template-columns: repeat(5, 1fr 2fr);
}
```

The implicit and explicit grid

When creating our example grid we specifically defined our column tracks with the `grid-template-columns` property, but the grid also created rows on its own. These rows are part of the implicit grid. Whereas the explicit grid consists of any rows and columns defined with `grid-template-columns` or `grid-template-rows`.

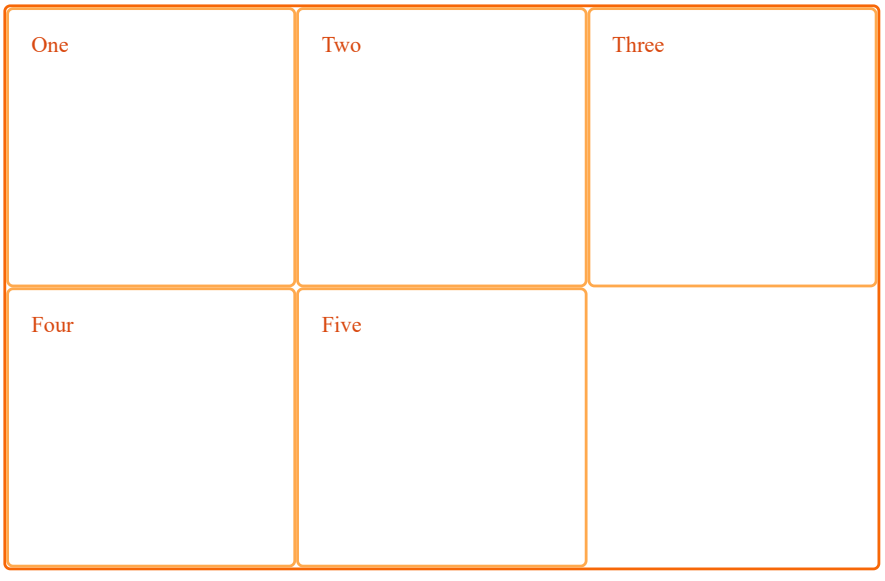
If you place something outside of the defined grid—or due to the amount of content, more grid tracks are needed—then the grid creates rows and columns in the implicit grid. These tracks will be auto-sized by default, resulting in their size being based on the content that is inside them.

You can also define a set size for tracks created in the implicit grid with the `grid-auto-rows` and `grid-auto-columns` properties.

In the below example we use `grid-auto-rows` to ensure that tracks created in the implicit grid are 200 pixels tall.

```
<div class="wrapper">
  <div>One</div>
  <div>Two</div>
  <div>Three</div>
  <div>Four</div>
  <div>Five</div>
</div>
```

```
.wrapper {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-auto-rows: 200px;
}
```



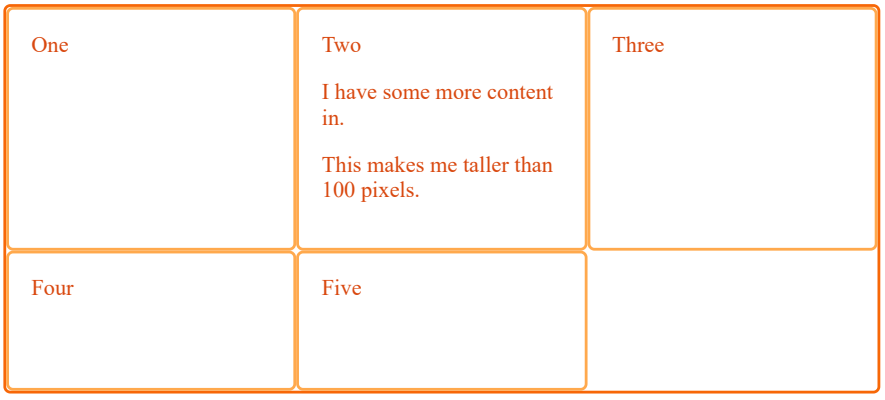
Track sizing and `minmax()`

When setting up an explicit grid or defining the sizing for automatically created rows or columns we may want to give tracks a minimum size, but also ensure they expand to fit any content that is added. For example, I may want my rows to never collapse smaller than 100 pixels, but if my content stretches to 300 pixels in height, then I would like the row to stretch to that height.

Grid has a solution for this with the `minmax()` function. In this next example I am using `minmax()` in the value of `grid-auto-rows`. This means automatically created rows will be a minimum of 100 pixels tall, and a maximum of `auto`. Using `auto` means that the size will look at the content size and will stretch to give space for the tallest item in a cell, in this row.

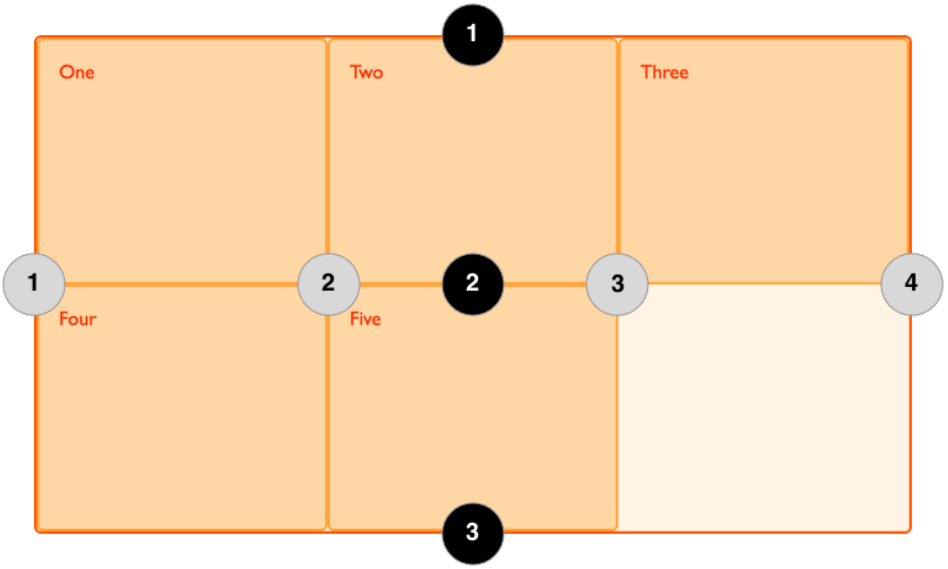
```
.wrapper {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-auto-rows: minmax(100px, auto);
}
```

```
<div class="wrapper">
  <div>One</div>
  <div>Two
    <p>I have some more content in.</p>
    <p>This makes me taller than 100 pixels.</p>
  </div>
  <div>Three</div>
  <div>Four</div>
  <div>Five</div>
</div>
```



Grid lines

It should be noted that when we define a grid we define the grid tracks, not the lines. Grid then gives us numbered lines to use when positioning items. In our three column, two row grid we have four column lines.



Lines are numbered according to the writing mode of the document. In a left-to-right language, line 1 is on the left-hand side of the grid. In a right-to-left language, it is on the right-hand side of the grid. Lines can also be named, and we will look at how to do this in a later guide in this series.

Positioning items against lines

We will be exploring line based placement in full detail in a later article. The following example demonstrates doing this in a simple way. When placing an item, we target the line – rather than the track.

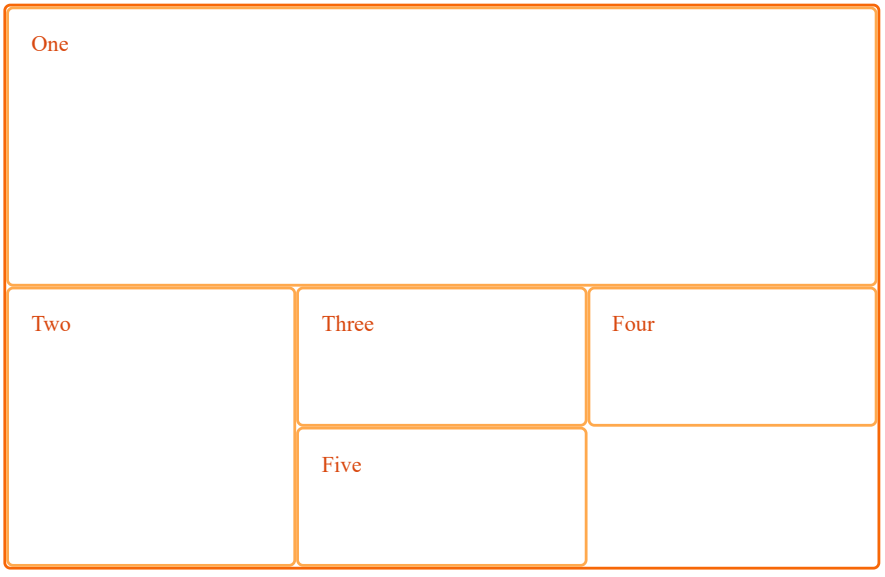
In the following example I am placing the first two items on our three column track grid, using the `grid-column-start`, `grid-column-end`, `grid-row-start` and `grid-row-end` properties. Working from left to right, the first item is placed against column line 1, and spans to column line 4, which in our case is the far-right line on the grid. It begins at row line 1 and ends at row line 3, therefore spanning two row tracks.

The second item starts on grid column line 1, and spans one track. This is the default so I do not need to specify the end line. It also spans two row tracks from row line 3 to row line 5. The other items will place themselves into empty spaces on the grid.

```
<div class="wrapper">
  <div class="box1">One</div>
  <div class="box2">Two</div>
  <div class="box3">Three</div>
  <div class="box4">Four</div>
  <div class="box5">Five</div>
</div>
```

```
.wrapper {
  display: grid;
```

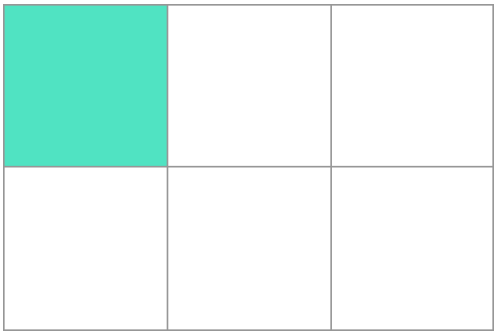
```
  grid-template-columns: repeat(3, 1fr);
  grid-auto-rows: 100px;
}

.box1 {
  grid-column-start: 1;
  grid-column-end: 4;
  grid-row-start: 1;
  grid-row-end: 3;
}

.box2 {
  grid-column-start: 1;
  grid-row-start: 3;
  grid-row-end: 5;
}
```



Don't forget that you can use the Grid Inspector in Firefox Developer Tools to see how the items are positioned against the lines of the grid.

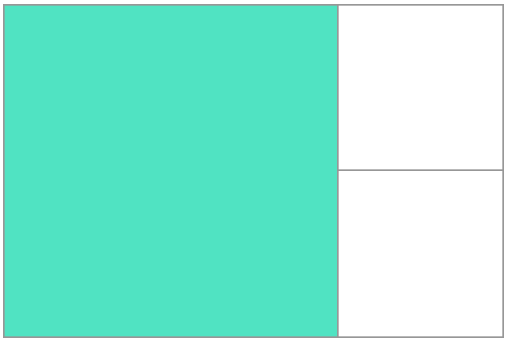
Grid cells

A *grid cell* is the smallest unit on a grid. Conceptually it is like a table cell. As we saw in our earlier examples, once a grid is defined as a parent the child items will lay themselves out in one cell each of the defined grid. In the below image, I have highlighted the first cell of the grid.



Grid areas

Items can span one or more cells both by row or by column, and this creates a *grid area*. Grid areas must be rectangular – it isn't possible to create an L-shaped area for example. The highlighted grid area spans two row and two column tracks.



Gutters

Gutters or *alleys* between grid cells can be created using the `column-gap` and `row-gap` properties, or the shorthand `gap`. In the below example, I am creating a 10-pixel gap between columns and a `1em` gap between rows.

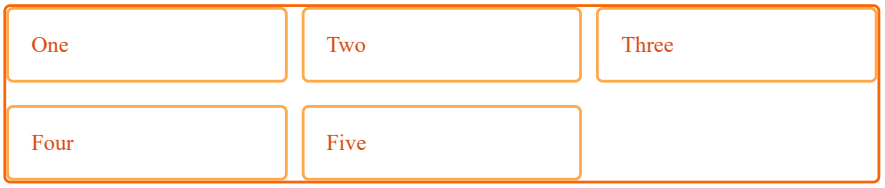
```
.wrapper {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  column-gap: 10px;
  row-gap: 1em;
}
```

Note: When grid first shipped in browsers the `column-gap`, `row-gap` and `gap` were prefixed with the `grid-` prefix as `grid-column-gap`, `grid-row-gap` and `grid-gap` respectively.

Browsers are updating to remove this prefix, however the prefixed versions will be maintained as aliases making them safe to use. At the present time, some browsers do not

support the unprefixed versions which is why the live examples in this guide use grid- prefixed versions.

```
<div class="wrapper">
  <div>One</div>
  <div>Two</div>
  <div>Three</div>
  <div>Four</div>
  <div>Five</div>
</div>
```



Any space used by gaps will be accounted for before space is assigned to the flexible length `fr` tracks, and gaps act for sizing purposes like a regular grid track, however you cannot place anything into a gap. In terms of line-based positioning, the gap acts like a fat line.

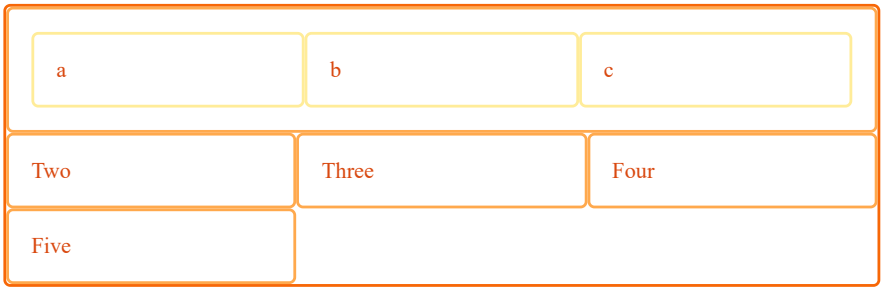
Nesting grids

A grid item can become a grid container. In the following example, I have the three-column grid that I created earlier, with our two positioned items. In this case the first item has some sub-items. As these items are not direct children of the grid they do not participate in grid layout and so display in a normal document flow.

```
<div class="wrapper">
  <div class="box box1">
    <div class="nested">a</div>
    <div class="nested">b</div>
    <div class="nested">c</div>
  </div>
  <div class="box box2">Two</div>
  <div class="box box3">Three</div>
  <div class="box box4">Four</div>
  <div class="box box5">Five</div>
</div>
```

If I set `box1` to `display: grid` I can give it a track definition and it too will become a grid. The items then lay out on this new grid.

```
.box1 {
  grid-column-start: 1;
  grid-column-end: 4;
  grid-row-start: 1;
  grid-row-end: 3;
  display: grid;
  grid-template-columns: repeat(3, 1fr);
}
```



In this case the nested grid has no relationship to the parent. As you can see in the example it has not inherited the `grid-gap` of the parent and the lines in the nested grid do not align to the lines in the parent grid.

Subgrid

In the working draft of the Level 2 Grid specification there is a feature called *subgrid*, which would let us create nested grids that use the track definition of the parent grid.

> **Note**: Subgrids are not yet implemented in any browsers, and the specification is subject to change.

In the current specification, we would edit the above nested grid example to change the track definition of `grid-template-columns: repeat(3, 1fr)`, to `grid-template-columns: subgrid`. The nested grid will then use the parent grid tracks to layout items.

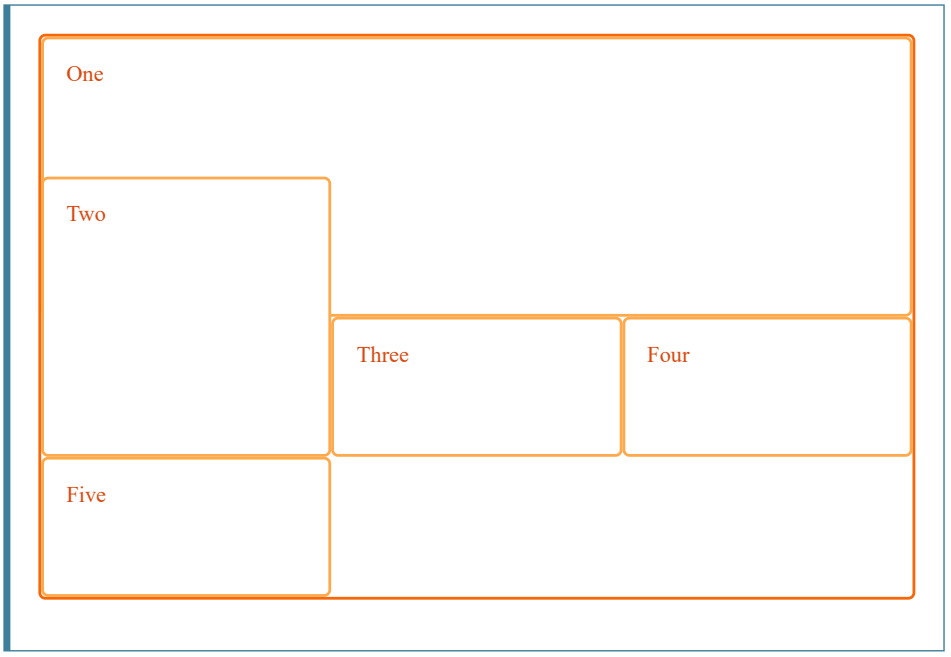
```
.box1 {
  grid-column-start: 1;
  grid-column-end: 4;
  grid-row-start: 1;
  grid-row-end: 3;
  display: grid;
  grid-template-columns: subgrid;
}
```

Layering items with `z-index`

Grid items can occupy the same cell. If we return to our example with items positioned by line number, we can change this to make two items overlap.

```
<div class="wrapper">
  <div class="box box1">One</div>
  <div class="box box2">Two</div>
  <div class="box box3">Three</div>
  <div class="box box4">Four</div>
  <div class="box box5">Five</div>
</div>
```

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

.box1 {
  grid-column-start: 1;
  grid-column-end: 4;
  grid-row-start: 1;
  grid-row-end: 3;
}

.box2 {
  grid-column-start: 1;
  grid-row-start: 2;
  grid-row-end: 4;
}
```

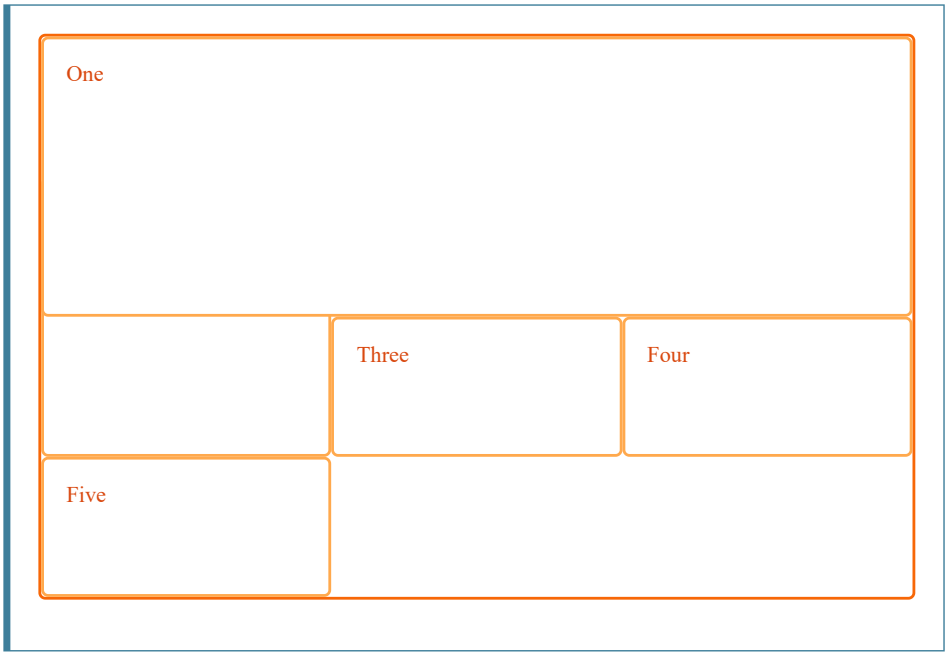


The item `box2` is now overlapping `box1`, it displays on top as it comes later in the source order.

Controlling the order

We can control the order in which items stack up by using the `z-index` property - just like positioned items. If we give `box2` a lower `z-index` than `box1` it will display below `box1` in the stack.

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

.box1 {
  grid-column-start: 1;
  grid-column-end: 4;
  grid-row-start: 1;
  grid-row-end: 3;
  z-index: 2;
}

.box2 {
  grid-column-start: 1;
  grid-row-start: 2;
  grid-row-end: 4;
  z-index: 1;
}
```



Next Steps

In this article we have had a very quick look through the Grid Layout Specification. Have a play with the code examples, and then move onto the next part of this guide where we will really start to dig into the detail of CSS Grid Layout.