2.4.8 Position the Grid Items

Let's redefine the grid dimensions by adding a set width and center to the page. Let's also add a grid gap of 10px:

```
.service-grid-container {
  max-width: 940px;
  margin: 0 auto;
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-template-rows: repeat(3, 100px);
  grid-gap: 10px;
}
```

IMPORTANT

> Please note that `grid-gap` is shorthand notation for the properties `grid-row-gap` and `grid-column-gap` and can be written together in one value or separately in two values as:
>
> ```
> .grid-container {
>  grid-gap: <grid-row-gap> <grid-column-gap>;
> }
> ```

> Where `<grid-row-gap>` can take in a static or relative unit size to declare a gap between rows. Conversely `<grid-column-gap>` declares the column gap. In the CSS rule above, the single value will be assigned to both column and row gaps. This is similar to setting one value for the `padding` property, which sets this value for all four sides of the `padding` in our CSS box model.

In the CSS rule for our grid items, let's remove the height and width declarations to allow the background color to fill the grid cell. Let's add a little padding as well and keep our font color black:

```
.service-grid-item {
  background-color: orange;
  border: solid;
  padding: 1em;
  color: black;
}
```

Now we can start placing our grid items into the grid by creating CSS class selectors that will uniquely identify each grid item and allow us to position each item individually. Until now, we've been using grid properties on the grid parent element or grid container to define the grid. In contrast, this step defines the grid properties on the grid children or grid items.

Because we can customize each grid item individually, we must use a class selector to target each grid item's HTML element by its class attribute. This is similar to the flexbox lesson, when we assigned properties to the parent element in order to have consistent behavior in the children elements—whether that be alignment in relation to the parent or spacing in relation to sibling elements— or when we assigned properties to the flexbox child element to control it individually for responsive behavior.

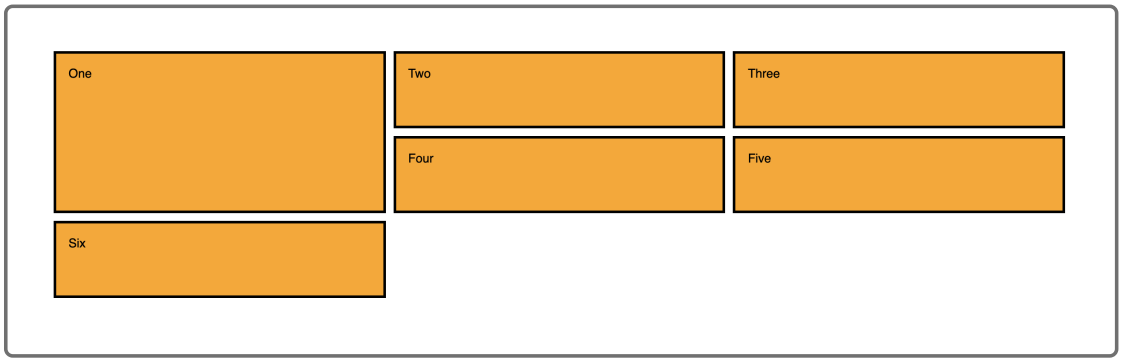
Let's start with a simple example by assigning unique classes to each grid item . We'll label them as box1, box2, etc.

```
<div class="service-grid-container">
  <div class="service-grid-item box1">One</div>
  <div class="service-grid-item box2">Two</div>
  <div class="service-grid-item box3">Three</div>
  <div class="service-grid-item box4">Four</div>
  <div class="service-grid-item box5">Five</div>
  <div class="service-grid-item box6">Six</div>
</div>
```

Next, we can create a CSS rule for the `box1` class selector and designate the size of this item by identifying the grid lines it starts and ends on:

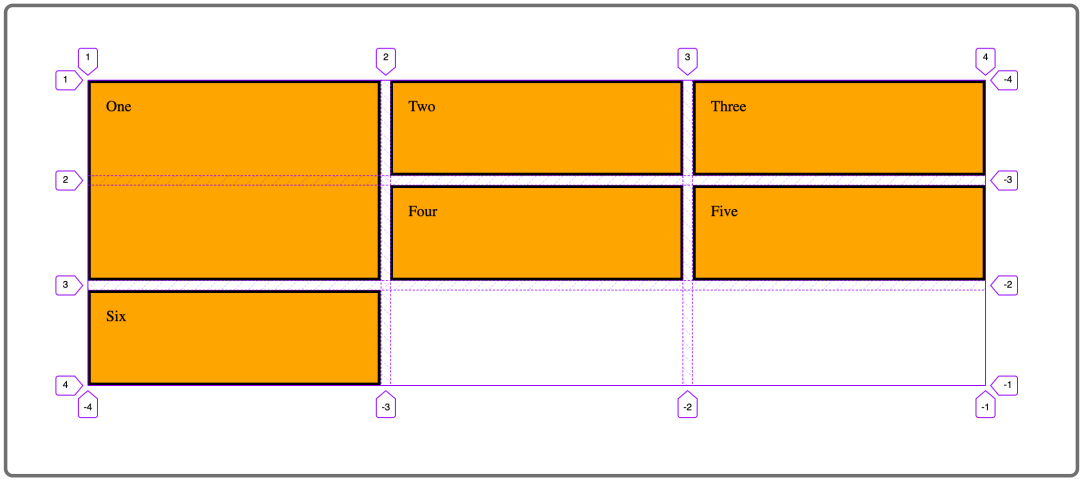
```
.box1 {
  grid-column-start: 1;
  grid-column-end: 2;
  grid-row-start: 1;
  grid-row-end: 3;
}
```

This should render the following in the browser:

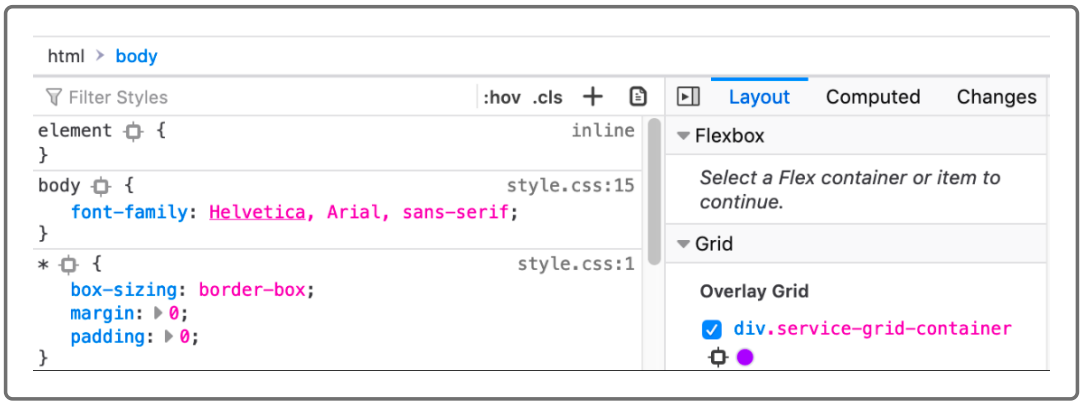


IMPORTANT

> If you want to view the grid line overlay, as shown here:



The Mozilla Firefox browser has a unique set of tools for CSS Grid. One that can be particularly helpful when positioning grid items is the Overlay Grid, which you can find in Firefox's Inspector window:



Notice how the grid item was able to span two rows by defining the row start and row end properties:

- With the first two declarations, `grid-column-start: 1` and `grid-column-end: 2`, we define the position and width of the grid track of this grid item to start on the first vertical grid line at the start or at the grid line 1 position.

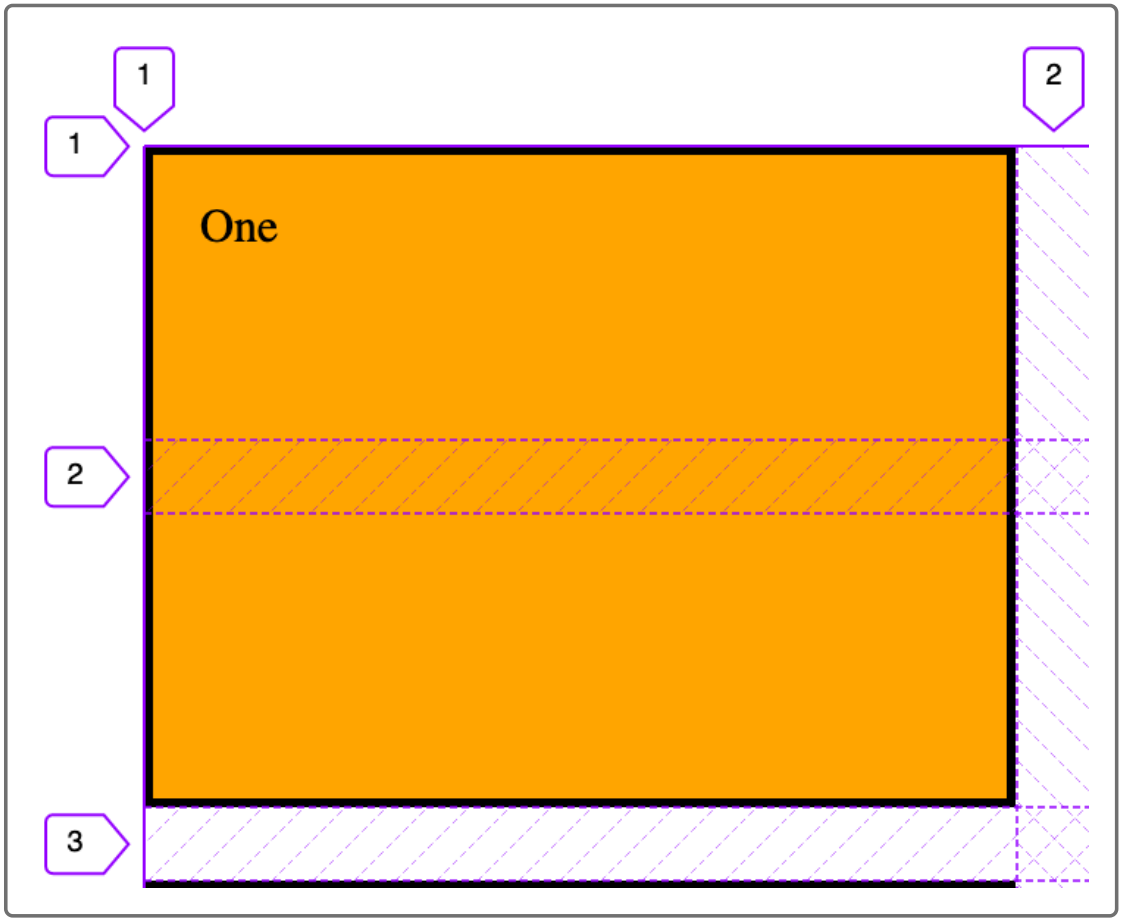
- With the `end` declaration, we define the grid track to span one column because the column will end on the adjacent grid line 2.

The result is that the width of the defined box1 will start from grid line 1 and end at grid line 2.

The following two declarations define the position and height of the grid item:

- `grid-row-start: 1` defines the box1 grid item to start at the top grid line
- `grid-row-end: 3` defines the box1 grid item to span two rows with the boundary defined to end on grid line 3

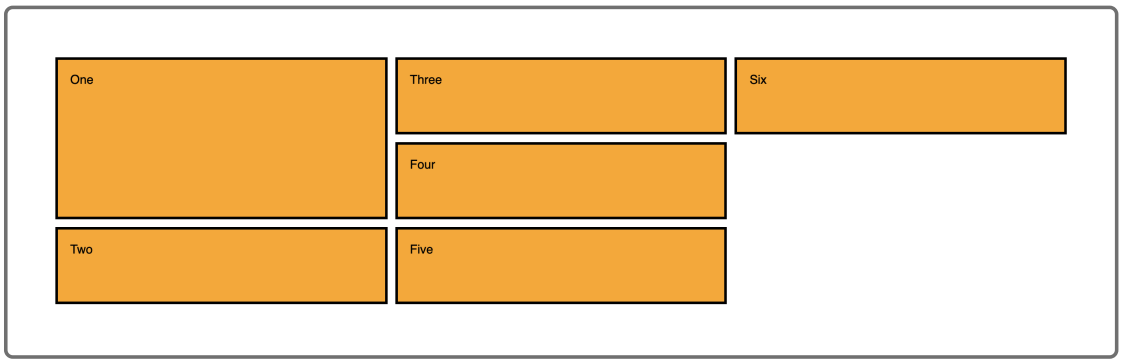
You can see the result here:



CSS Grid automatically positioned the remaining grid children by filling each row before wrapping to the following row. This default property comes from `grid-auto-flow`, which was set to `row`.

Let's temporarily change this property to `column` by adding the CSS declaration, `grid-auto-flow: column;`, to `.service-grid-container` for

the following effect:



Notice how the remaining grid items now fill the grid by column. Let's remove the `grid-auto-flow: column;` declaration for now to reset the default back to the property value of `row` for the purposes of our lesson.

There's also a shortcut notation when writing our position declarations using grid lines. Our box1 was originally written as follows:

```
.box1 {
  grid-column-start: 1;
  grid-column-end: 2;
  grid-row-start: 1;
  grid-row-end: 3;
}
```

Using the `grid-column` and `grid-row` properties allows the start and end values to be assigned in one declaration:

```
.box1 {
  grid-column: 1/2;
  grid-row: 1/3;
}
```

The value of these properties can also be stated using the negative number designations:

```
.box1 {
  grid-column: 1/-3;
  grid-row: 1/-2;
}
```

This is especially useful in the case where the grid is incredibly large because the end of the grid will be -1.

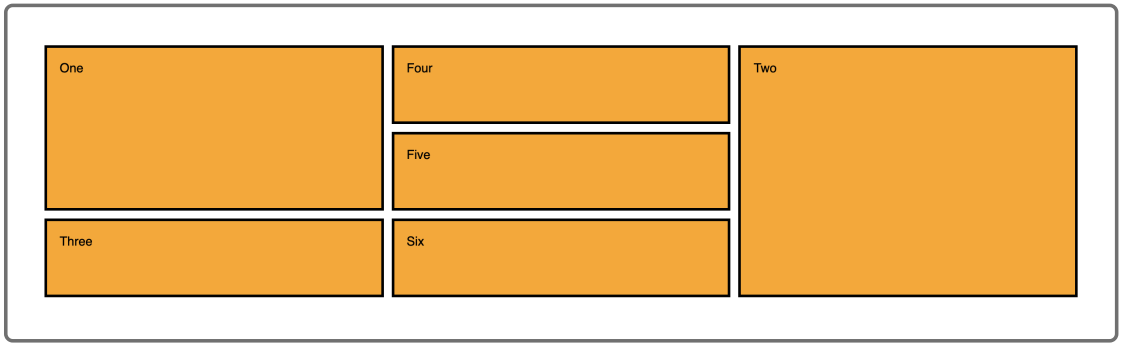
The values of the `grid-column` and `grid-row` properties can even be assigned by explicitly stating how many grid cells are spanned:

```
.box1 {
  grid-column: 1/ span 1;
  grid-row: 1/ span 2;
}
```

Let's add a few more CSS rules for some more grid items so we can familiarize ourselves with how to position and size grid items:

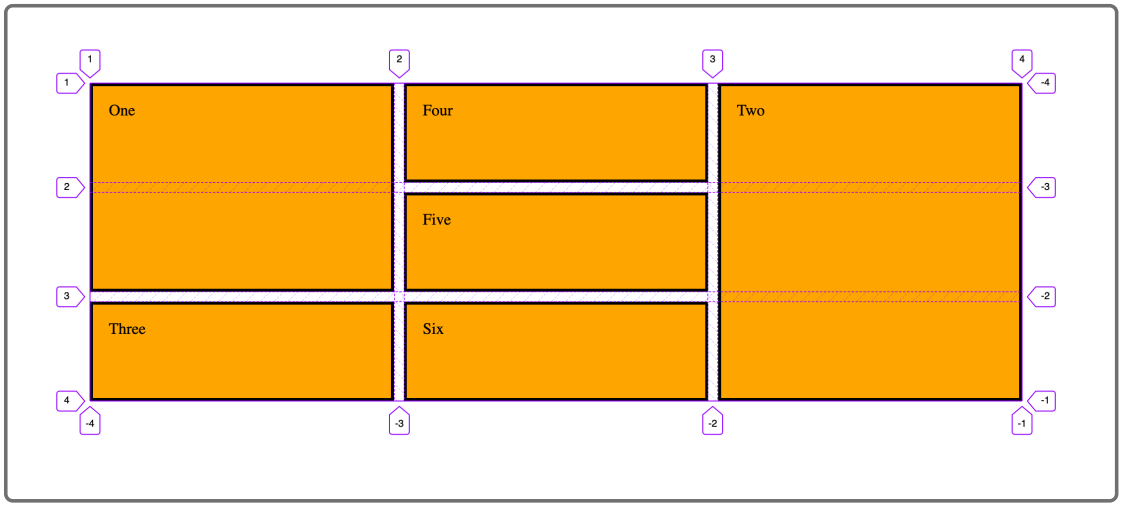
```
 .box2 {
  grid-column: 3/4;
  grid-row: 1/-1;
}
```

Save the file and render in the browser:



As you can see, box2 has been removed from the normal grid flow and assigned a position in the third column with the declaration `grid-column: 3/4;`. Box2 spans three rows because we declared the box to span from the top grid line to the last grid line with the declaration `grid-row: 1/-1;`.

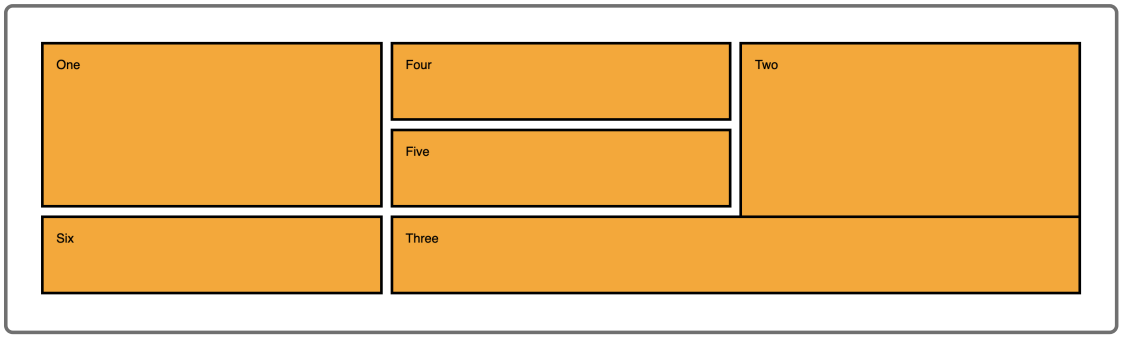
Here's the grid with the grid line overlay:



Now what happens if we position a box that overlaps?

```
.box3 {
  grid-column: 2/ span 2;
  grid-row: 3/ span 1;
}
```

In this rule, we're declaring that box3 will span two columns and sit in the third row. Save the file and render in the browser:



As you can see, box6 moved into the first column due to the `grid-auto-flow` property, and box3 now overlaps box2.

DEEP DIVE ▲

DEEP DIVE

CSS Grid is a robust layout model with many properties that can be used for complex positioning. To learn more, please see the following resources:

- **MDN web docs on grid-area (https://developer.mozilla.org/en-US/docs/Web/CSS/grid-area)**
- **MDN web docs on grid-auto-columns (https://developer.mozilla.org/en-US/docs/Web/CSS/grid-auto-columns)**