2.4.6 Get Started with CSS Grid

So what is CSS Grid? Much like it sounds, CSS Grid is a set of vertical and horizontal lines that intersect much like a table. Elements can be positioned in the grid between the vertical or horizontal lines.

To create a grid, we first define the container element and the width and number of columns and rows. Then we declare the height and width of the child elements that sit inside the grid.

REWIND

Many of the CSS Grid terms and parent/child relationships are similar to flexbox.

Create the Grid Container

A **grid container** is declared by using `display: grid` and will act as the wrapper for the grid. The direct children of the grid container are the **grid items**. These elements sit within the vertical and horizontal **grid lines**.

Let's begin by writing some basic HTML that will establish our grid container and grid items. In `index.html`, add the following element right before the `<section id="reach-out" class="contact">` element:

```
<section id="service-plans" class="services">
</section>
```

Then add the link in the `<nav>`:

```
<a href="#service-plans">Service Plans</a>
```

Within the `<section>` element, add some example HTML to start the grid by adding the grid container with its class attribute and the grid items with their corresponding class attributes:

```
<div class="service-grid-container">
  <div class="service-grid-item">One</div>
  <div class="service-grid-item">Two</div>
  <div class="service-grid-item">Three</div>
  <div class="service-grid-item">Four</div>
  <div class="service-grid-item">Five</div>
  <div class="service-grid-item">Six</div>
</div>
```

In `style.css`, add the CSS rule to set up the grid:

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

```
}
```

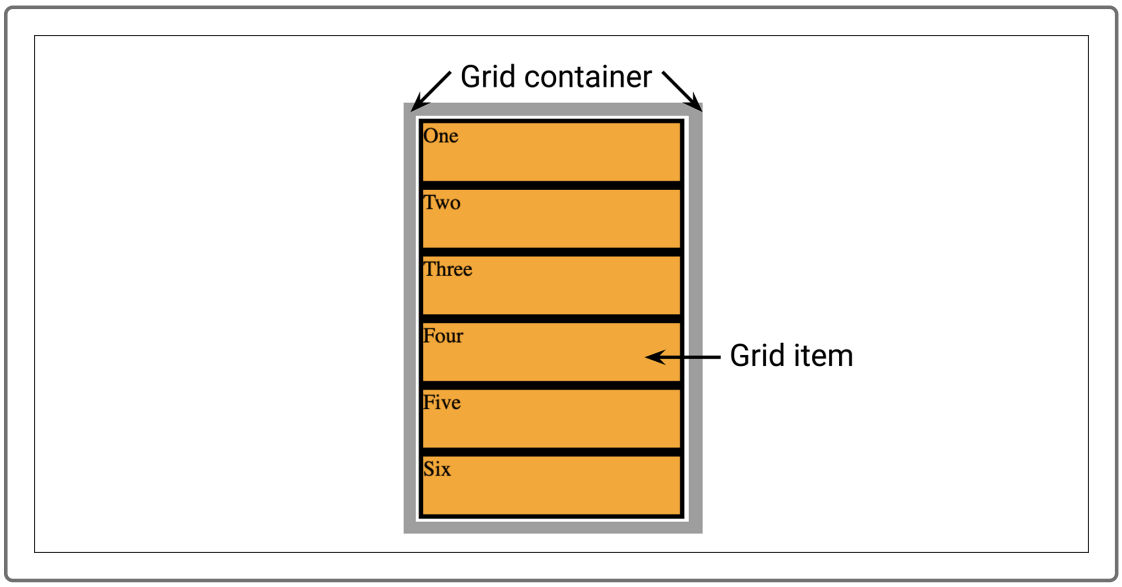
IMPORTANT

> Make sure these new CSS rules are added BEFORE the media queries. Later on, we will adjust the grid for mobile devices in the media queries, but those overrides will only work if the media queries are defined last in the style sheet.

Now add some styling to the grid items to help visualize our example:

```
.service-grid-item {
  width: 200px;
  height: 50px;
  background-color: orange;
  border: solid;
  color: black;
}
```

Save the files and render the grid in the browser:



Great job! We'll make sure this grid has the correct number of rows and columns in the next step.

DEEP DIVE ▲

DEEP DIVE

`inline-grid` and `subgrid` are also possible values of the `display` property. Here's how these differ:

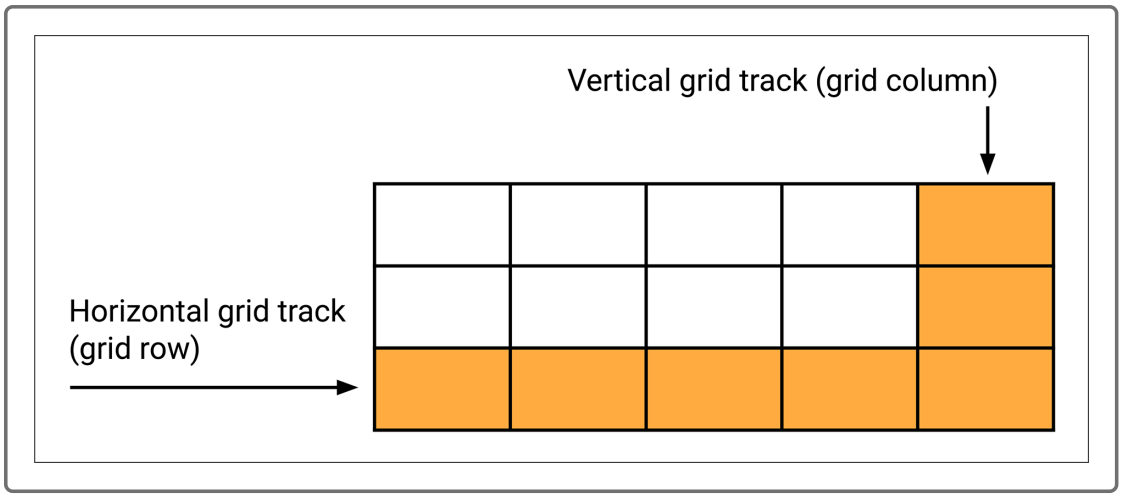
- `grid`: Creates a block-level grid container, which means it will sit alone in its own row.
- `inline-grid`: Creates a inline-level grid container, which means it will sit besides other elements in the same row according to available space and document flow.
- `subgrid` (https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout/Subgrid): Creates a grid container on a grid item within a grid container. To learn more, check out the **MDN web docs about CSS Grid Layout** (https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout).

Now pause to add and commit your work in the feature branch. The commit message should be related to the work you just completed, such as "added grid example."

Format the Rows and Columns

A **grid column** is defined by the space between two adjacent vertical lines. The size of the column is determined by the `grid-template-columns` property. Conversely, a **grid row** is defined by the space between two adjacent horizontal lines whose height is determined by the `grid-template-rows` property.

A **grid track** is a generic term for either a grid column or grid row, shown here:



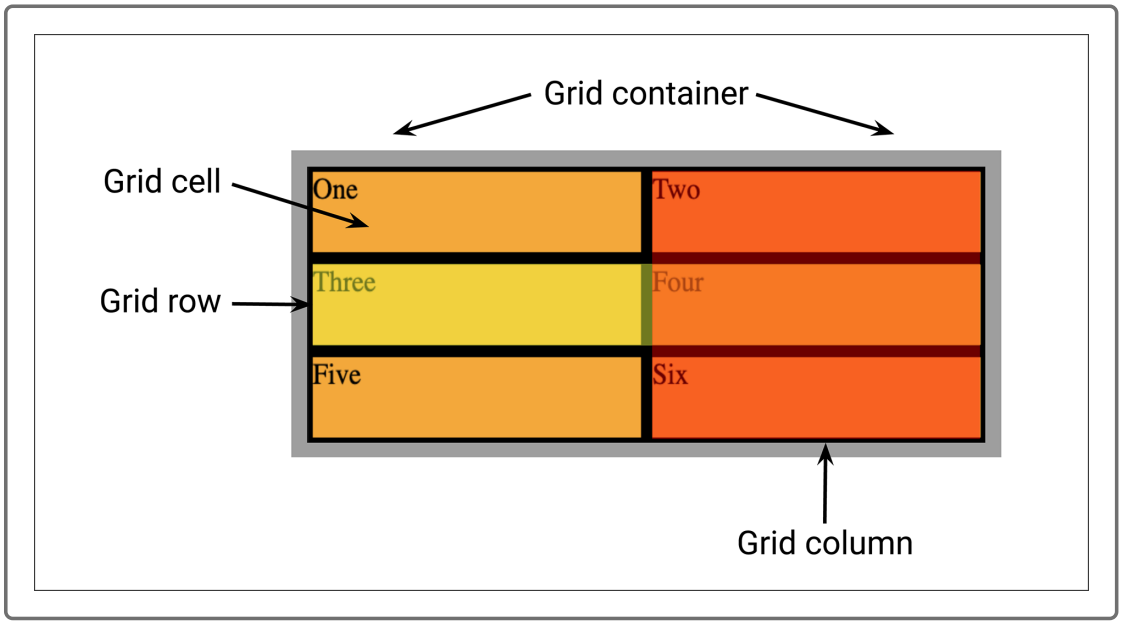
To establish the dimensions of the grid, declare either the `grid-template-columns` or `grid-template-rows`. Let's add the declaration of the `grid-template-columns` property to the `.service-grid-container` rule:

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

In the `grid-template-columns` declaration, we determined the number and width of grid columns. The number of columns is determined by the

number of values (in this case, there are two columns because `200px` appears twice).

Here's how the grid renders now:



In the image above, a single orange box defined as the area between adjacent vertical lines and horizontal lines represents a **grid cell**.

Try it yourself: experiment with width sizes and the number of values and observe how the grid changes. If you set the column width to 150px or smaller, the column widths will no longer be consistent.

Making the column width skinnier than the size of our HTML element, the `<div>`, will truncate the element except for the last cells on the end of the row. These ending grid cells are allowed to overlap and not hide the overflow due to the absence of an adjacent element, so they will span to 200px, the size of the element.

Now let's add the grid row declaration to the grid container's CSS rule:

```
.service-grid-container {
  display: grid;
  grid-template-columns: 200px 200px;
```

```
  grid-template-rows: 50px 50px 50px;
}
```

If we compare these two in the browser, we can see that they look identical. This is because in our first example when we only declared the `grid-template-columns`, CSS Grid was able to determine the number of rows and row heights needed based on the height of the content (50px).

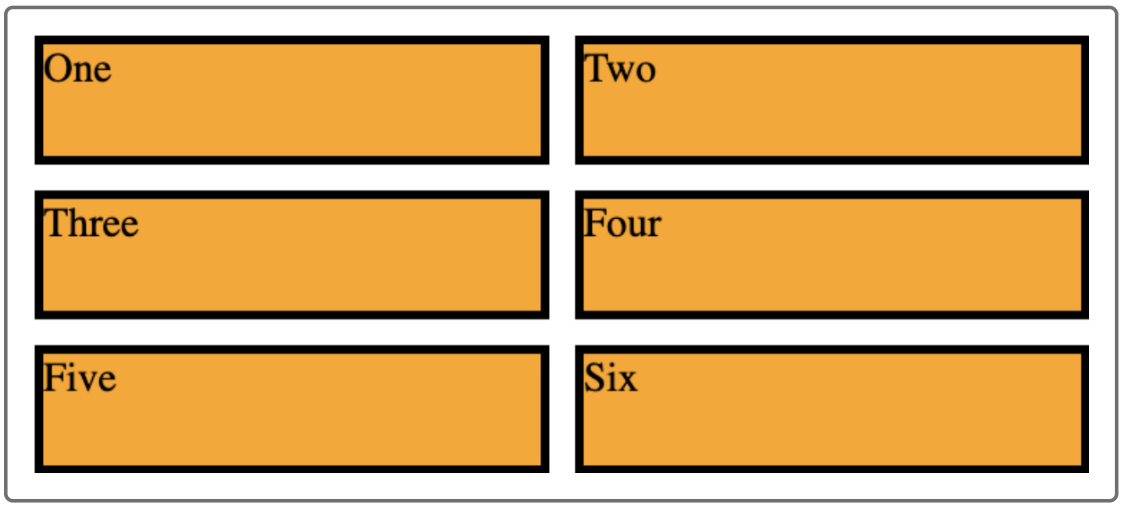
This is called the **implicit grid** due to the auto-generation of rows by CSS Grid. When we declared the rows with the `grid-template-rows` declaration, we created an **explicit grid** because we declared all the rows needed for our content.

What if we increase the row height and column width by changing the declaration to the following?

```
.service-grid-container {
  display: grid;
  grid-template-columns: 210px 210px;
  grid-template-rows: 60px 60px 60px;
}
```

We can see that a gap was generated in between grid items to fill in the extra space because the `<div>` element's height stops at 50px and its width stops at 200px. Due to the default `box-sizing` property value set to `border-content` of the grid, gaps were only created within the grid cells and not in the margins surrounding the grid container.

You can explicitly create these gaps by using the property `grid-gap` on the grid parent element:



Congrats! You've created your first CSS grid. Let's continue learning about relative units we can assign to the `grid-template-columns` and `grid-template-rows` properties to create a responsive layout.

Create a Responsive Grid

Currently, our grid is static. This means the column, row, and cell sizes do not change regardless of the viewport or screen size.

To create a responsive layout, we can assign relative units to the `grid-template-columns` and `grid-template-rows` properties to allow flexible grid tracks. We can do this by using some of the values we previously covered, namely the `percent` or `rem` unit values.

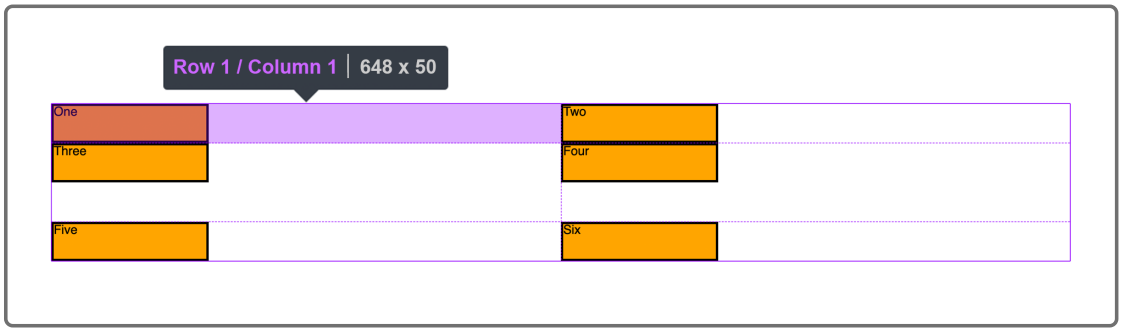
CSS Grid also introduces a relative unit length value called `fr` or **fraction**. This relative value is unique to CSS Grid and cannot be used elsewhere. It's called fraction because CSS Grid assigns the unit length based on a fraction of the available space.

Let's use `fr` in an example:

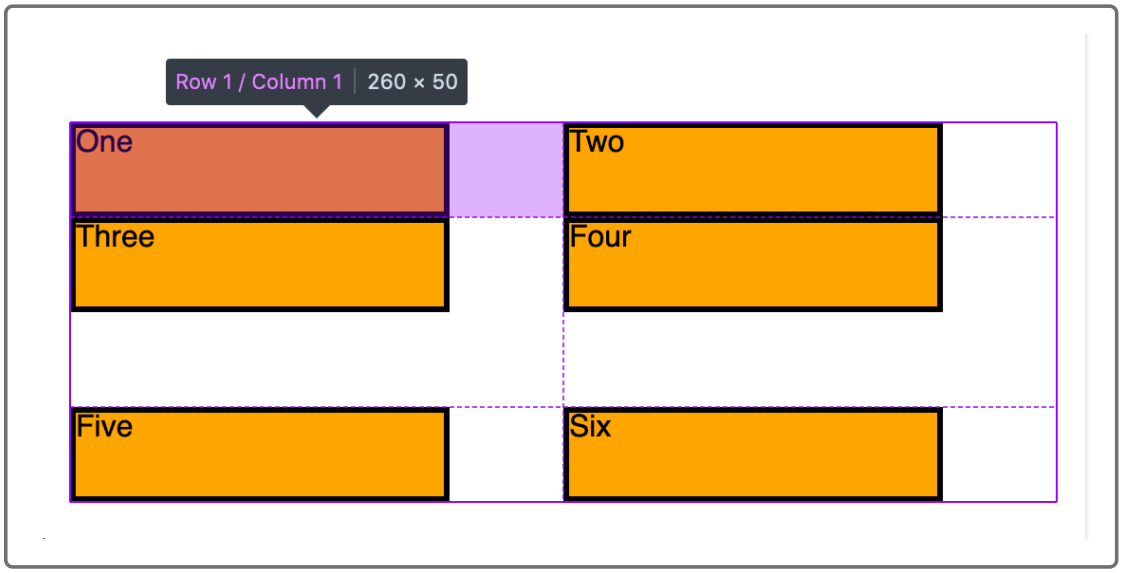
```
.service-grid-container {
  display: grid;
  grid-template-columns: 1fr 1fr;
```

```
  grid-template-rows: 1fr 2fr 1fr;
}
```

The following image demonstrates how the available space was distributed between the two columns:



As you can see in the `grid-template-columns` declaration, the `1fr 1fr` value creates two equal-width tracks that grow and shrink depending on the available space. Because our grid container is a block-level element and doesn't have a defined width, the entire screen was set as the width. When we reduce the screen width to 500px, the columns remain equal in size as the `fr` unit evenly distributes the available space, as shown here:



In the `grid-template-rows` declaration, we set the value to `1fr 2fr 1fr`, which caused the second row to be twice the height as the other rows. In cases when there is no "available" space, a `fr` unit will be determined by CSS Grid to be the height of the content in the row with the most height. Two `fr` units in this row example will be twice this height.

Using the repeat() Notation

When dealing with a large grid that has many adjacent rows or columns with the same values, we can use the `repeat` notation to identify the values of the row and column grid declarations.

The following examples explore two rules that render identically; the second example uses `repeat`.

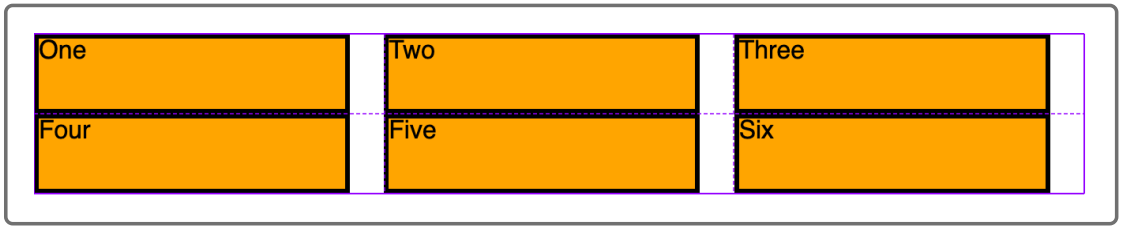
Here's the first example:

```
.service-grid-container {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
  grid-template-rows: 1fr 1fr;
}
```

The next example uses the `repeat()` notation:

```
.service-grid-container {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-template-rows: repeat(2, 1fr);
}
```

Here's how this renders:



Notice in the `repeat()` notation parentheses, the number of columns or rows is followed by a comma and then the width of the column or row. We can also use different units of length (such as px, rem, or %) or use it with other values to note other track sizes.

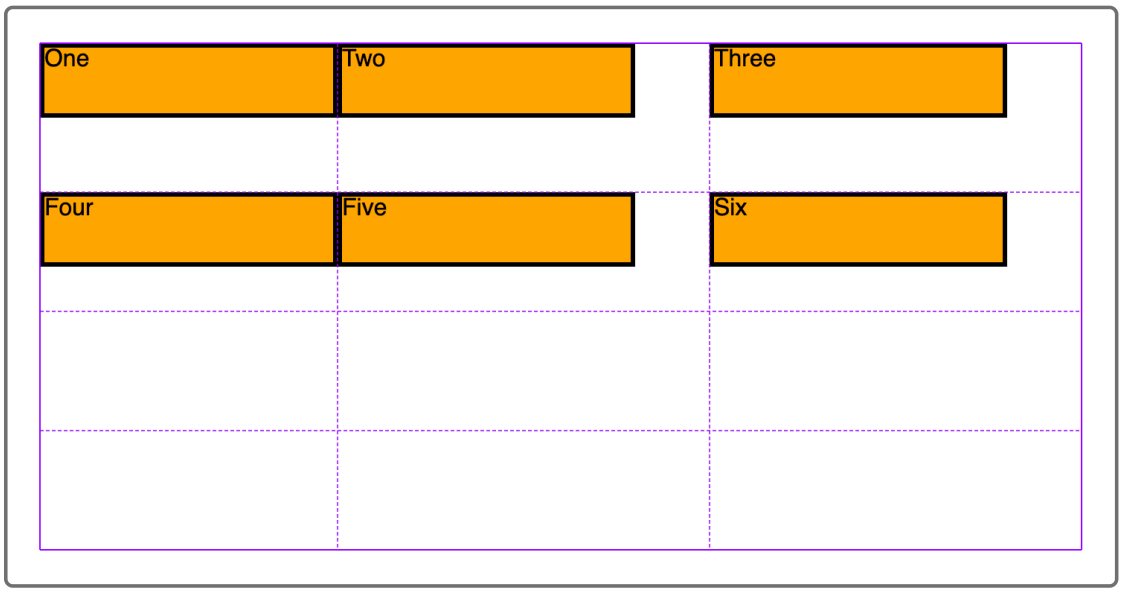
Here's an example:

```
.service-grid-container {
  display: grid;
  grid-template-columns: 150px repeat(2, 1fr);
  grid-template-rows: repeat(4, 5rem);
}
```

The above code is equivalent to the following rule:

```
.service-grid-container {
  display: grid;
  grid-template-columns: 150px 1fr 1fr;
  grid-template-rows: 5rem 5rem 5rem 5rem;
}
```

Let's take a look at how this rule is rendered by the browser:



PAUSE

What is the large gap of space located at the bottom the picture?

The explicit grid, as defined in our `.service-grid-container` rule, created four rows even though there is only enough content for two.

Hide Answer

Because CSS Grid is built to create complex responsive 2D layouts, it shouldn't be surprising to hear that there is a vast number of unique grid properties and property values that help achieve customization and efficiency. Here are a few of the most popular and useful ones to remember:

- **minmax()**: Offers a range of possible values with a minimum value and a maximum value. This can be set as a column or row size value designating the possible size range for the width or height of the grid track. An example is `minmax(100px, 1fr)`.

- **auto-fit**: Often used in combination with `minmax()`; allows CSS Grid to calculate the number of columns or rows to accommodate the grid items.

- **auto**: A property value that uses CSS Grid to calculate the size of the column or row.

Try playing with different combinations of these values to see how they affect the grid. For instance, you could try the following options:

```
.service-grid-container {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(100px, 1fr));
  grid-template-rows: repeat(4, minmax(25px, auto));
}
```

This type of auto-fit scenario would work well for a large number of photos needing to automatically fit in a photo gallery.

Take a moment now to assess your understanding with the following knowledge check:

Which CSS rule would create a grid that is 3x4?

```
.service-grid-container {
  display: grid;
  grid-template-columns: 1fr repeat(3, 150px);
  grid-template-rows: 100px repeat(4, 1fr);
}
```

```
.service-grid-container {
  display: grid;
  grid-template-columns: 1fr repeat(2, 2fr);
  grid-template-rows: 2fr repeat(2, 40px) 1fr;
}
```

```
.service-grid-container {
  display: grid;
  grid-template-columns: 1fr repeat(2, 2fr);
  grid-template-rows: 2fr repeat(2, 1fr);
}
```

```
.service-grid-container {
  display: grid;
  grid-template-columns: repeat(3, 250px) 3fr;
  grid-template-rows: 2fr repeat(2, 50px) 1fr;
}
```

Check Answer

Finish ▸

© 2020 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.