Grid:

The *grid* can be used to lay out entire web pages. Whereas Flexbox is mostly useful for positioning items in a one-dimensional layout, CSS grid is most useful for two-dimensional layouts, providing many tools for aligning and moving elements across both rows and columns.

To turn an HTML element into a grid container, you must set the element's displayproperty to grid (for a block-level grid) or inline-grid (for an inline grid). Then, you can assign other properties to lay out the grid.

Creating columns:

By default, grids contain only one column. If you were to start adding items, each item would be put on a new row; that's not much of a grid! To change this, we need to explicitly define the number of rows and columns in our grid.

We can define the columns of our grid by using the CSS property grid-template-columns. Below is an example of this property in action:

```
.grid {
  display: grid;
  width: 500px;
  grid-template-columns: 100px 200px;
}
```

This property creates two changes. First, it defines the number of columns in the grid; in this case, there are two. Second, it sets the width of each column. The first column will be 100 pixels wide and the second column will be 200 pixels wide.

We can also define the size of our columns as a percentage of the entire grid's width.

```
.grid {
  display: grid;
  width: 1000px;
  grid-template-columns: 20% 50%;
}
```

In this example, the grid is 1000 pixels wide. Therefore, the first column will be 200 pixels wide because it is set to be 20% of the grid's width. The second column will be 500 pixels wide.

We can also mix and match these two units. In the example below, there are three columns of width 20 pixels, 40 pixels, and 60 pixels:

```
.grid {
  display: grid;
  width: 100px;
  grid-template-columns: 20px 40% 60px;
}
```

Notice that in this example, the total width of our columns (120 pixels) exceeds the width of the grid (100 pixels). This might make our grid cover other elements on the page! In a later exercise we will discuss how to avoid overflow.

Creating Rows

We've learned how to define the number of columns in our grid explicitly. To specify the number and size of the rows, we are going to use the property grid-template-rows. This property is almost identical to grid-template-columns. Take a look at the code below to see both properties in action.

```
.grid {
  display: grid;
  width: 1000px;
  height: 500px;
  grid-template-columns: 100px 200px;
  grid-template-rows: 10% 20% 600px;
}
```

This grid has two columns and three rows. grid-template-rows defines the number of rows and sets each row's height. In this example, the first row is 50 pixels tall (10% of 500), the second row is 100 pixels tall (20% of 500), and the third row is 600 pixels tall.

Grid Template

The property grid-template can replace the previous two CSS properties. Both grid-template-rows and grid-template-columns are nowhere to be found in the following code!

```
.grid {
  display: grid;
  width: 1000px;
  height: 500px;
  grid-template: 200px 300px / 20% 10% 70%;
}
```

When using grid-template, the values before the slash will determine the size of each row. The values after the slash determine the size of each column. In this example, we've made two rows and three columns of varying sizes.

Fraction

You may already be familiar with several types of responsive units such as percentages (%), ems and rems. CSS Grid introduced a new relative sizing unit — fr, like fraction. By using the fr unit, we can define the size of columns and rows as a fraction of the grid's length and width. This unit was specifically created for use in CSS Grid. Using fr makes it easier to prevent grid items from overflowing the boundaries of the grid. Consider the code below:

```
.grid {
  display: grid;
  width: 1000px;
  height: 400px;
  grid-template: 2fr 1fr 1fr / 1fr 3fr 1fr;
}
```

In this example, the grid will have three rows and three columns. The rows are splitting up the available 400 pixels of height into four parts. The first row gets two of those parts, the second column gets one, and the third column gets one. Therefore the first row is 200 pixels tall, and the second and third rows are 100 pixels tall.

Each column's width is a fraction of the available space. In this case, the available space is split into five parts. The first column gets one-fifth of the space, the second column gets three-fifths, and the last column gets one-fifth. Since the total width is 1000 pixels, this means that the columns will have widths of 200 pixels, 600 pixels, and 200 pixels respectively.

Repeat

The properties that define the number of rows and columns in a grid can take a function as a value. repeat() is one of these functions. The repeat() function was created specifically for CSS Grid.

```
.grid {
  display: grid;
  width: 300px;
  grid-template-columns: repeat(3, 100px);
}
```

minmax

So far, all of the grids that we have worked with have been a fixed size. The grid in our example has been 400 pixels wide and 500 pixels tall. But sometimes you might want a grid to resize based on the size of your web browser.

In these situations, you might want to prevent a row or column from getting too big or too small. For example, if you have a 100-pixel wide image in your grid, you probably don't want its column to get thinner than 100 pixels! The minmax() function can help us solve this problem.

```
.grid {
  display: grid;
  grid-template-columns: 100px minmax(100px, 500px) 100px;
}
```

In this example, the first and third columns will always be 100 pixels wide, no matter the size of the grid. The second column, however, will vary in size as the overall grid resizes. The second column will always be between 100 and 500 pixels wide.

Grid Gap

In all of our grids so far, there hasn't been any space between the items in our grid. The CSS properties grid-row-gap and grid-column-gapwill put blank space between every row and column in the grid.

```
.grid {
  display: grid;
  width: 320px;
  grid-template-columns: repeat(3, 1fr);
  grid-column-gap: 10px;
}
```

It is important to note that `grid-gap` does not add space at the beginning or end of the grid. In the example code, our grid will have three columns with two ten-pixel gaps between them.

Let's quickly calculate how wide these columns are. Remember that using `fr`considers all of the *available* space. The grid is 320 pixels wide and 20 of those pixels are taken up by the two grid gaps. Therefore each column takes a piece of the 300 available pixels. Each column gets `1fr`, so the columns are evenly divided into thirds (or 100 pixels each).

Finally, there is a CSS property `grid-gap` that can set the row and column gap at the same time. `grid-gap: 20px 10px;` will set the distance between rows to 20 pixels and the distance between columns to 10 pixels. Unlike other CSS grid properties, this shorthand does not take a `/` between values! If only one value is given, it will set the column gap and the row gap to that value.

Grid Items

In this lesson, we have learned how to define a grid container. When explicitly defining a grid, you have to declare the quantity of rows and columns and their respective sizes.
In all of our examples, the items placed in the grid have always taken up exactly one square. This does not always need to be the case; we can drastically change the look of our grid by making grid items take up more than one row and one column.

Multiple Row Items

Using the CSS properties `grid-row-start` and `grid-row-end`, we can make single grid items take up multiple rows. Remember, we are no longer applying CSS to the outer grid container; we're adding CSS to the elements sitting inside the grid!

```
.item {
  grid-row-start: 1;
  grid-row-end: 3;
}
```

In this example, the HTML element of class `item` will take up two rows in the grid, rows 1 and 2.

The value for `grid-row-start` should be the row at which you want the grid item to begin. The value for `grid-row-end` should be one greater than the row at which you want the grid item to end. An element that covers rows 2, 3, and 4 should have these declarations:`grid-row-start: 2` and `grid-row-end: 5`.

We can use the property `grid-row` as shorthand for `grid-row-start` and `grid-row-end`. The following two code blocks will produce the same output:

```
.item {
  grid-row-start: 4;
  grid-row-end: 6;
}
.item {
  grid-row: 4 / 6;
}
```

Grid Column

The previous three properties also exist for columns. `grid-column-start`, `grid-column-end` and `grid-column` work identically to the row properties. These properties allow a grid item to span multiple columns.

When using these properties, we can use the keyword `span` to start or end a column or row relative to its other end. Look at how `span` is used in the code below:

```
.item {
  grid-column: 4 / span 2;
}
```

This is telling the item element to begin in column four and take up two columns of space. So item would occupy columns four and five. It produces the same result as the following code blocks:

```
.item {
  grid-column: 4 / 6;
}
```

Grid Area

We've already been able to use grid-row and grid-column as shorthand for properties like grid-row-start and grid-row-end. We can refactor even more using the property grid-area. This property will set the starting and ending positions for both the rows and columns of an item.

```
.item {
  grid-area: 2 / 3 / 4 / span 5;
}
```

grid-area takes four values separated by slashes. The order is important! This is how grid-area will interpret those values.

1. grid-row-start
2. grid-column-start
3. grid-row-end
4. Grid-column-end

Overlapping elements

Another powerful feature of CSS Grid Layout is the ability to easily overlap elements. When overlapping elements, it is generally easiest to use grid line names and the grid-area property.

```
<style>
.container {
  display: grid;
  max-width: 900px;
  position: relative;
  margin: auto;
  grid-gap: 10px;
  grid-template: repeat(12, 1fr) / repeat(6, 1fr);
}

h1, h2, h3 {
  font-family: monospace;
  text-align: center;
}

header {
  background-color: dodgerblue;
  grid-area: 1 / 1 / 3 / 7;
}

nav {
  background-color: beige;
  grid-area: 3 / 1 / 4 / 7;
}

.left {
  background-color: dodgerblue;
  grid-area: 4 /1 / 9 / 5;

}

.right {
  background-color: beige;
  grid-area: 4 / 5 / 9 / 7;
}

.overlap {
  background-color: lightcoral;
  grid-area: 6 / 5 / 8 / 7;
  z-index: 5;
}

footer {
  background-color: dodgerblue;
  grid-area: 9 / 1 / 13 / 7;
}
</style>
  <div class="container">
    <header>
      <h1>Header</h1>
    </header>
    <nav>
      <h1>Nav</h1>
    </nav>
    <section class="left">
      <h2>Left</h2>
    </section>
    <div class="overlap">
      <h3>Overlap!</h3>
    </div>
    <section class="right">
      <h2>Right</h2>
    </section>
    <footer>
      <h1>Footer</h1>
    </footer>
  </div>
```

Justify items:

We have referred to "two-dimensional grid-based layout" several times throughout this course.

There are two axes in a grid layout — the *column* (or block) axis and the *row* (or inline) axis.

The column axis stretches from top to bottom across the web page.

The row axis stretches from left to right across the web page.

`justify-items` is a property that positions grid items along the inline, or row, axis. This means that it positions items from left to right across the web page.

justify-items accepts these values:

- start — aligns grid items to the left side of the grid area
- end — aligns grid items to the right side of the grid area
- center — aligns grid items to the center of the grid area
- stretch — stretches all items to fill the grid area

Justify Content

In the previous exercise, we learned how to position elements within their columns. In this exercise, we will learn how to position a grid within its parent element.

We can use justify-content to position the entire grid along the row axis.

It accepts these values:

- start — aligns the grid to the left side of the grid container
- end — aligns the grid to the right side of the grid container
- center — centers the grid horizontally in the grid container
- stretch — stretches the grid items to increase the size of the grid to expand horizontally across the container
- space-around — includes an equal amount of space on each side of a grid element, resulting in double the amount of space between elements as there is before the first and after the last element
- space-between — includes an equal amount of space between grid items and no space at either end
- space-evenly — places an even amount of space between grid items and at either end

Align Items

In the previous two exercises, we learned how to position grid items and grid columns from left to right across the page. Below, we'll learn how to position grid items from top to bottom!

`align-items` is a property that positions grid items along the block, or column axis. This means that it positions items from top to bottom.

`align-items` accepts these values:

- `start` — aligns grid items to the top side of the grid area
- `end` — aligns grid items to the bottom side of the grid area
- `center` — aligns grid items to the center of the grid area
- `stretch` — stretches all items to fill the grid area

Justify Self and Align Self

The `justify-items` and `align-items` properties specify how all grid items contained within a single container will position themselves along the row and column axes, respectively.

`justify-self` specifies how an individual element should position itself with respect to the row axis. This property will override `justify-items` for any item on which it is declared.

`align-self` specifies how an individual element should position itself with respect to the column axis. This property will override `align-items` for any item on which it is declared.

They both accept these four properties:

- `start` — positions grid items on the left side/top of the grid area
- `end` — positions grid items on the right side/bottom of the grid area
- `center` — positions grid items on the center of the grid area
- `stretch` — positions grid items to fill the grid area (default)

`align-self` and `justify-self` accept the same values as `align-items` and `justify-items`. You can read about these values on the Mozilla Developer Network.

Ex:

```
main {
  height: 1600px;
  display: grid;
  grid-template-columns: 250px 250px;
  grid-template-rows: repeat(3, 450px);
  grid-gap: 20px;
  margin-top: 44px;
  background-color:grey;
  justify-items: center;
  justify-content: center;
  align-items: stretch;
  align-content: center;
  border:1px solid red;
}

h2 {
  font-family: Poppins;
  font-size: 18px;
  font-weight: 600;
  letter-spacing: 0.3px;
  text-align: left;
  color: #ffa500;
  padding: 10px 0px 10px 10px;
}

img {
  width: 100%;
  height: auto;
}

.recipe {
  box-shadow: 0 1px 1px 0 rgba(0, 0, 0, 0.5);
  width:200px;
  border:1px solid blue;
}

.a {
  background-color:white;
  border:1px solid red;
}

.b {
}
.c {
  background-color: brown;
}

.time {
  padding-left: 10px;
  padding-top: 10px;
  width: 20px;
  height: auto;
}

.mins {
  display: inline-block;
  font-family: Poppins;
  font-size: 14px;
  font-weight: 500;
  letter-spacing: 0.3px;
  text-align: left;
  color: #4a4a4a;
  position: relative;
  bottom: 5px;
}

.description {
  font-family: Work Sans;
  font-size: 14px;
  font-weight: 300;
  line-height: 1.29;
  letter-spacing: 0.1px;
  text-align: left;
  color: #4a4a4a;
  padding: 10px 0px 10px 10px;
  border-top: 1px solid #4a4a4a;
}

.logo {
  width: 115px;
  height: 21.1px;
  object-fit: contain;
  padding: 20px;
}

.container {
  min-width: 500px;
  margin: auto;
}


<main>

        <div class="recipe a">
          <img
src="https://s3.amazonaws.com/codecademy-content/courses/learn-css-grid/lesson-ii/i
mage-1.png" />
          <h2>CHOCOLATE MOUSSE</h2>
          <img
src="https://s3.amazonaws.com/codecademy-content/courses/learn-css-grid/lesson-ii/ti
me.svg" class="time" />
          <p class="mins">20 mins</p>
          <p class="description">
            This delicious chocolate mousse will delight dinner guests of all ages!</p>
        </div>

        <div class="recipe b">
          <img
src="https://s3.amazonaws.com/codecademy-content/courses/learn-css-grid/lesson-ii/i
mage-2.png" />
          <h2>SMOKED LAMB WITH RICE</h2>
          <img
src="https://s3.amazonaws.com/codecademy-content/courses/learn-css-grid/lesson-ii/ti
me.svg" class="time" />
          <p class="mins">120 mins</p>
          <p class="description">
            Want to feel like your favorite relative came over and made you dinner? This
comfort meal of smoked lamb and rice will quickly become a weekend favorite!
          </p>
        </div>

        <div class="recipe c">
          <img
src="https://s3.amazonaws.com/codecademy-content/courses/learn-css-grid/lesson-ii/i
mage-5.png" />
          <h2>GOAT CHEESE SALAD</h2>
          <img
src="https://s3.amazonaws.com/codecademy-content/courses/learn-css-grid/lesson-ii/ti
me.svg" class="time" />
          <p class="mins">25 mins</p>
          <p class="description">
            In addition to the full flavor of goat cheese, this salad includes kale, avocado,
and farro to balance it out.</p>
        </div>

        <div class="recipe d">
          <img
src="https://s3.amazonaws.com/codecademy-content/courses/learn-css-grid/lesson-ii/i
mage-4.png" />
          <h2>CHICKEN SANDWICH</h2>
          <img
src="https://s3.amazonaws.com/codecademy-content/courses/learn-css-grid/lesson-ii/ti
me.svg" class="time" />
          <p class="mins">45 mins</p>
          <p class="description">
            We've packed a lot into this one - shredded cabbage, carmalized onions,
deep-fried chicken, chipotle mayo, half-sour pickles, and a toasted sesame bun will
leave you thoroughly satisfied!</p>
        </div>

        <div class="recipe e">
          <img
src="https://s3.amazonaws.com/codecademy-content/courses/learn-css-grid/lesson-ii/i
mage-3.png" />
          <h2>SWEET CHURROS</h2>
          <img
src="https://s3.amazonaws.com/codecademy-content/courses/learn-css-grid/lesson-ii/ti
me.svg" class="time" />
```

```
        <p class="mins">90 mins</p>
        <p class="description">
          Making this classic summer treat at home will remind you of a childhood spent in
the park.</p>
      </div>

    </main>
```

Implicit vs. Explicit Grid

So far, we have been explicitly defining the dimensions and quantities of our grid elements using various properties. This works well in many cases, such as a landing page for a business that will display a specific amount of information at all times. However, there are instances in which we don't know how much information we're going to display. For example, consider online shopping. Often, these web pages include the option at the bottom of the search results to display a certain quantity of results or to display ALL results on a single page. When displaying all results, the web developer can't know in advance how many elements will be in the search results each time.
What happens if the developer has specified a 3-column, 5-row grid (for a total of 15 items), but the search results return 30?

Something called the *implicit* grid takes over. The implicit grid is an algorithm built into the specification for CSS Grid that determines default behavior for the placement of elements when there are more than fit into the grid specified by the CSS.
The default behavior of the implicit grid is as follows: items fill up rows first, adding new rows as necessary.

```
<body>
  <div>Part 1</div>
  <div>Part 2</div>
  <div>Part 3</div>
  <div>Part 4</div>
  <div>Part 5</div>
</body>

body {
  display: grid;
  grid: repeat(2, 100px) / repeat(2, 150px);
  grid-auto-rows: 50px;
}
```

In the example above, there are 5 `<div>`s. However, in the `section` rule set, we only specify a 2-row, 2-column grid — four grid cells.

The fifth `<div>` will be added to an implicit row that will be 50 pixels tall.

If we did not specify `grid-auto-rows`, the rows would be auto-adjusted to the height of the content of the grid items.

Grid Auto Flow

In addition to setting the dimensions of implicitly-added rows and columns, we can specify the order in which they are rendered.

`grid-auto-flow` specifies whether new elements should be added to rows or columns.

`grid-auto-flow` accepts these values:

- `row` — specifies the new elements should fill rows from left to right and create new rows when there are too many elements (default)
- `column` — specifies the new elements should fill columns from top to bottom and create new columns when there are too many elements
- `dense` — this keyword invokes an algorithm that attempts to fill holes earlier in the grid layout if smaller elements are added

You can pair `row` and `column` with `dense`, like this: `grid-auto-flow: row dense;`

Ex:

```
<style>
main {
  display: grid;
  grid-template-columns: 250px 250px;
  grid-template-rows: repeat(3, 450px);
  grid-auto-rows: 500px;
  grid-auto-flow: row;
  grid-gap: 20px;
  margin-top: 44px;
  background-color:grey;
  justify-items: center;
  justify-content: center;
  align-items: stretch;
  align-content: center;
  border:1px solid red;
}

h2 {
  font-family: Poppins;
  font-size: 18px;
  font-weight: 600;
  letter-spacing: 0.3px;
  text-align: left;
  color: #ffa500;
  padding: 10px 0px 10px 10px;
}

img {
  width: 100%;
  height: auto;
}

.recipe {
  box-shadow: 0 1px 1px 0 rgba(0, 0, 0, 0.5);
  width:200px;
  border:1px solid blue;
}
```

```css
.a {
  background-color:white;
  border:1px solid red;
}

.b {

}
.c {
  background-color: brown;
}

.time {
  padding-left: 10px;
  padding-top: 10px;
  width: 20px;
  height: auto;
}

.mins {
  display: inline-block;
  font-family: Poppins;
  font-size: 14px;
  font-weight: 500;
  letter-spacing: 0.3px;
  text-align: left;
  color: #4a4a4a;
  position: relative;
  bottom: 5px;
}

.description {
  font-family: Work Sans;
  font-size: 14px;
  font-weight: 300;
  line-height: 1.29;
  letter-spacing: 0.1px;
  text-align: left;
  color: #4a4a4a;
  padding: 10px 0px 10px 10px;
  border-top: 1px solid #4a4a4a;
}

.logo {
  width: 115px;
  height: 21.1px;
  object-fit: contain;
  padding: 20px;
}

.container {
  min-width: 500px;
  margin: auto;
}
</style>
<main>

        <div class="recipe a">
          <img
src="https://s3.amazonaws.com/codecademy-content/courses/learn-css-grid/lesson-ii/i
mage-1.png" />
          <h2>CHOCOLATE MOUSSE</h2>
          <img
src="https://s3.amazonaws.com/codecademy-content/courses/learn-css-grid/lesson-ii/ti
me.svg" class="time" />
          <p class="mins">20 mins</p>
          <p class="description">
            This delicious chocolate mousse will delight dinner guests of all ages!</p>
        </div>

        <div class="recipe b">
          <img
src="https://s3.amazonaws.com/codecademy-content/courses/learn-css-grid/lesson-ii/i
mage-2.png" />
          <h2>SMOKED LAMB WITH RICE</h2>
          <img
src="https://s3.amazonaws.com/codecademy-content/courses/learn-css-grid/lesson-ii/ti
me.svg" class="time" />
```

```
        <p class="mins">120 mins</p>
        <p class="description">
          Want to feel like your favorite relative came over and made you dinner? This
comfort meal of smoked lamb and rice will quickly become a weekend favorite!
        </p>
      </div>
      <div class="recipe c">
        <img
src="https://s3.amazonaws.com/codecademy-content/courses/learn-css-grid/lesson-ii/i
mage-5.png" />
        <h2>GOAT CHEESE SALAD</h2>
        <img
src="https://s3.amazonaws.com/codecademy-content/courses/learn-css-grid/lesson-ii/ti
me.svg" class="time" />
        <p class="mins">25 mins</p>
        <p class="description">
          In addition to the full flavor of goat cheese, this salad includes kale, avocado,
and farro to balance it out.</p>
      </div>

      <div class="recipe d">
        <img
src="https://s3.amazonaws.com/codecademy-content/courses/learn-css-grid/lesson-ii/i
mage-4.png" />
        <h2>CHICKEN SANDWICH</h2>
        <img
src="https://s3.amazonaws.com/codecademy-content/courses/learn-css-grid/lesson-ii/ti
me.svg" class="time" />
        <p class="mins">45 mins</p>
        <p class="description">
          We've packed a lot into this one - shredded cabbage, carmalized onions,
deep-fried chicken, chipotle mayo, half-sour pickles, and a toasted sesame bun will
leave you thoroughly satisfied!</p>
      </div>

      <div class="recipe e">
        <img
src="https://s3.amazonaws.com/codecademy-content/courses/learn-css-grid/lesson-ii/i
mage-3.png" />
```

```
        <h2>SWEET CHURROS</h2>
        <img
src="https://s3.amazonaws.com/codecademy-content/courses/learn-css-grid/lesson-ii/ti
me.svg" class="time" />
        <p class="mins">90 mins</p>
        <p class="description">
          Making this classic summer treat at home will remind you of a childhood spent in
the park.</p>
      </div>

      <div class="recipe e">
        <img
src="https://s3.amazonaws.com/codecademy-content/courses/learn-css-grid/lesson-ii/i
mage-3.png" />
        <h2>SWEET CHURROS</h2>
        <img
src="https://s3.amazonaws.com/codecademy-content/courses/learn-css-grid/lesson-ii/ti
me.svg" class="time" />
        <p class="mins">90 mins</p>
        <p class="description">
          Making this classic summer treat at home will remind you of a childhood spent in
the park.</p>
      </div>

      <div class="recipe e">
        <img
src="https://s3.amazonaws.com/codecademy-content/courses/learn-css-grid/lesson-ii/i
mage-3.png" />
        <h2>SWEET CHURROS</h2>
        <img
src="https://s3.amazonaws.com/codecademy-content/courses/learn-css-grid/lesson-ii/ti
me.svg" class="time" />
        <p class="mins">90 mins</p>
        <p class="description">
          Making this classic summer treat at home will remind you of a childhood spent in
the park.</p>
      </div>

      <div class="recipe e">
```

```
        <img
src="https://s3.amazonaws.com/codecademy-content/courses/learn-css-grid/lesson-ii/i
mage-3.png" />
        <h2>SWEET CHURROS</h2>
        <img
src="https://s3.amazonaws.com/codecademy-content/courses/learn-css-grid/lesson-ii/ti
me.svg" class="time" />
        <p class="mins">90 mins</p>
        <p class="description">
          Making this classic summer treat at home will remind you of a childhood spent in
the park.</p>
      </div>

    </main>
```
