1.

First, let’s take a look at our index.html file. As you can see, we have several <div>s that contain the content of our website.

Find the <div> with the class grid — this is our container grid and the properties applied to it will determine the structure for the content inside.

Next, locate the <div>s that share the class box and the <div>s that share the class testimonial. Their positioning will change the most when the new grid properties are added.

You can check this step off when you feel like you’re ready to proceed!

2.

Now let’s look at our CSS. We have provided the initial styling. Now it is up to you to add the necessary CSS grid properties to make sure the content is properly laid out on the page. The first step will be to initialize the grid, and then we’re going to specify the number and size of our rows and columns.

In style.css, inside the .grid ruleset, set the display property to grid.

3.

Next, to specify the number and size of the rows, inside the .grid ruleset, use the grid-template-rows property with a value of: 100px 8fr 5fr 4fr 5fr 80px. What changes do you see?

4.

Still inside of the .grid ruleset, set grid-template-columns to six equal sections using the fr measurement.

When you run your code, don’t panic! The CSS is just trying to organize our content based off of our instructions. In a later section, we’ll go over how to style our <div>s so they take up the necessary column widths.

5.

The code you added in the last two steps can be refactored into one shorthand grid-template property.

First, add the grid-template property to the .grid ruleset.

Next, cut the values from the grid-template-rows and grid-templates-columns properties, and paste them as values for the new grid-template property.

Then, delete the old grid-template-rows and grid-templates-columns declarations.

Finally, the grid-templates-columns value can be refactored even further. Use the repeat() function to refactor the 6 1fr values.

6.

Now, let’s fix that broken page layout. The content in the following rulesets needs to extend across all six columns:

header

.banner

.about

footer

Add the grid-column-start and grid-column-end properties to these rulesets, with values that start the content at the the first column and end it at the sixth column.

7.

The grid-column-start and grid-column-end properties can also be written as a shorthand property. Using the grid-column property and the span keyword, make:

.address span the first 2 columns

.hours span the 3rd and 4th columns

.call-us span the 5th and 6th columns

8.

In the .one and .two rulesets, use grid-column again to have each <div> span three columns.

9.

Let’s give our content some wiggle room. Return to the .grid ruleset, add a gap property, and set its value to 20 pixels. Notice which parts of the page have changed.

10.

Lastly, the <p> elements inside of the grid boxes aren’t centered. There are some fun ways to deal with this using some advanced CSS grid properties that you’ll soon learn about. In the meantime, one way we can take care of it with knowledge we already have is to make each of the boxes its own grid! Since each <p> element has a margin: auto declaration, they will be centered inside grid container.

To turn the boxes with the <p> elements into grids, use the display properties in the below rulesets to turn them into grids:

header

.about

.box

.testimonial

footer