Grid

Learning Objectives

- Explain the use cases for Grid.
- Understand and use CSS Grid syntax.
- Use CSS Grid to create a page layout.

What is CSS Grid Layout?

From the www.w3.org website...

"Grid Layout is a new layout model for CSS that has powerful abilities to control the sizing and positioning of boxes and their contents. Unlike Flexible Box Layout, which is single-axis–oriented, Grid Layout is optimized for 2-dimensional layouts: those in which alignment of content is desired in both dimensions."

With Grid layout, you can divide up the screen into `rows` and `columns` of sizes of your choosing, and then specify how many rows and columns each `cell` takes up. Sizings can be fixed, or dynamic, allowing you to create modern looking, versatile websites.
*Example of **flexbox** layout*

flex layout example
*example of **grid** layout*

grid layout example
Notice how the grid layout allows the cell on the right to take up multiple rows.

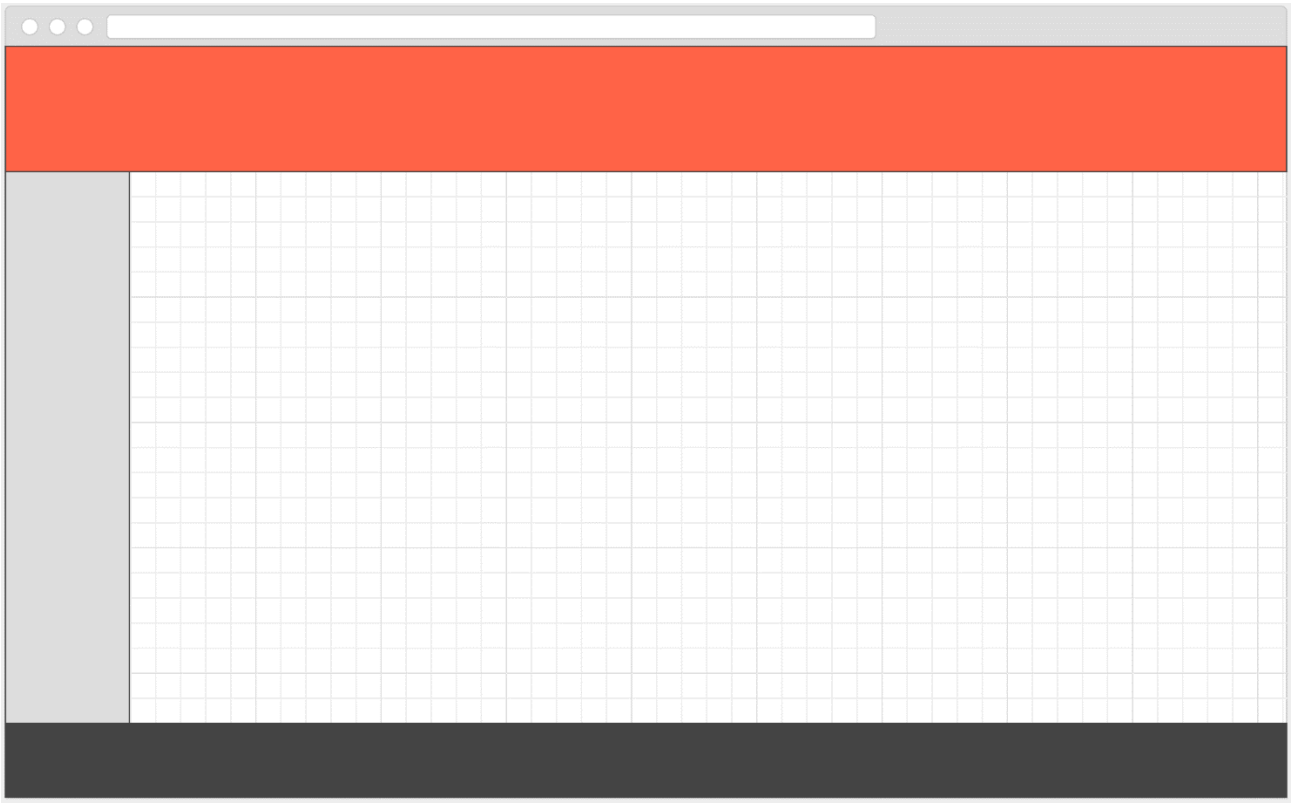
All the pieces of a Grid

When you look at a grid you see that it has **grid rows** and **grid columns**. Inside these rows and columns are **grid cells**. **Grid tracks** are one or more cells in a line in one rown or column. Easy, right?

The next concept is **grid areas**. Any group of adjacent grid cells can be sectioned into a **grid area**. We have the ability to give these areas names like we give our variables and link them through styles to our HTML containers.

What we'll be making

We will implement this basic web page template:



basic web template

The code for the Grid

We start with defining a container by setting the `display` property to `grid`. Since we are making a whole page layout we will use the `body` as our container:

```
body {
  display: grid;
  min-height: 100%;
}
```

All of our rows and columns will be inside this. We'll make it fill the full vertical space of the screen.

```
<body>
  <header></header>
  <aside></aside>
  <main></main>
  <footer></footer>
</body>
```

Now looking at the layout, if you extend all lines to the outermost container boundary, you will see the structure of the grid. Imagine that the left-center section was the middle part of a column that went from top to bottom. In that structure, the header and footer actually span two columns. We will see how to do that but first we must set up the rows and columns.

Code for the rows and columns

Our layout has 3 rows and 2 columns. We should note that the header and footer have a fixed height and the main fills the remaining space between them. Likewise, the aside will have a fixed width and the main will fill the rest of the space to the right.

First, we'll define the rows and columns:

```
body {

  display: grid;

  min-height: 100%;

  grid-template-rows: 200px 1fr 120px;

  grid-template-columns: 180px 1fr;

}
```

The `fr` is a **fractional unit** and saying `1fr` results in that area in the grid taking up 100% of the remaining space.

Linking HTML containers to Grid Areas

We have our header, aside, main, and footer and each will be mapped to one **grid area**. Here is how we do that:

```
header {

  grid-area: header;

}
```

```
aside {

  grid-area: sidebar;

}

main {

  grid-area: main;

}

footer {

  grid-area: footer:

}
```

We do not put quotes around these identifiers.

Laying out the Grid Areas

We can now add the CSS to layout our grid areas in the grid. There are a couple different ways of doing this but the way I like the most is using the `grid-template-areas` property because it gives you a nice way of visualizing your areas on top of the grid. Observe:

```
body {

  ...

  grid-template-areas: "header header"

                       "sidebar main"
```

```
                        "footer footer";


}
```

That looks truly bizarre, doesn't it? The syntax is definitely bizarre but it is perfectly legal and allows us to visually represent our grid in code. Rows are represented by each of those strings and columns are shown inside the row strings separated by spaces. Finish it off with a semicolon.

Getting a little Flexy with Grid

Grid itself can do some of the same vertical and horizontal spacing stuff that Flexbox can do. It uses the properties `justify-items` and `align-items`. But it is important to realize that it is only the thing marked with `display: grid` that has these abilities. The **grid areas** do not implcitly have any of that so if you need flex functionality in any of your grid areas, it's best to mark them as `display: flex` or `display: grid` and then apply your justification to the items therein.

Resources

- The CSS `grid` Module
- Wes Bos Teaches CSS-Grid
- Learn CSS Grid
  Screencasts

- Part 1
- Part 2