Responsive Design Tutorial Exercise

In this tutorial, we're going to walk through building a favorite tweets screen that will be responsive at many different screen sizes. We want to lay these items out in a grid-like fashion but also allow as many items on the screen as the creator of the webpage wishes to add.

You'll see in the `index.html` file, that we already have the basics of the screen laid out. There is a `header` at the top that has a logo and some navigation, a title underneath, some navigation, the twitter cards, and finally a footer at the bottom. All of these element are currently set up for mobile viewing, each in its own row, stacked vertically.

But as we make the screen bigger, the site doesn't use the bigger size very well at all. So we're going to fix this.

First, open the `index.html` file in VS Code and click the `Go Live` button. This will open a browser and show the page.

![Non responsive page](img/Screenshot_2019-02-26 Responsive Design Tutorial Exercise.png)

We can look at this page using different screen sizes within your browser. If you open the Developer Tools in your browser, there will be a button to put the browser in "Responsive Design Mode". You can also press Ctrl+Shift+M or Command+Shift+M to go into Responsive Design Mode.

This mode will have a couple of new features. You can change the screen size of the browser window and you can choose different device sizes from the drop down at the top.

Change the width of the screen to be 400 wide. This is roughly the size of a standard mobile phone and you can see that the site looks halfway decent there.

![Mobile phone view](img/Screenshot_2019-02-28 Responsive Design Tutorial Exercise.png)

As we expand this view to something like 725, or the iPad's 768, the elements start looking rather stretched on the screen. We can relayout these elements to make better use of the space at this width, so let's do that.

If you open the `style.css` file, you'll see our CSS defined for this page. Many of these styles are just look and feel rules, but one to take a look at is the rule for the grid layout of the site:

```
/* define the full page grid */
body {
  display: grid;
  grid-template-columns: 1em auto 1em;
  grid-template-areas:
      '. header .'
      '. nav .'
      '. content .'
      '. footer .';
}
```

```
header {
  grid-area: header;
}

nav {
  grid-area: nav;
}

main {
  grid-area: content;
}

footer {
  grid-area: footer;
}
```

Currently, the grid is defined for a small screen. There is one main column with a `1em` gutter down the right and left sides.

The navigation is set up like this:

```
/* Organize Nav Items */
nav ul {
  display: flex;
  flex-direction: column;
  justify-content: space-between;
  width: 100%;
}
```

This will flow the navigation in a column with each button taking the full width of the screen.

The tweets are also set up in a flex container with their `flex-basis`, or their set width in the container, set to `100%`.

```
main .container {
  display: flex;
  flex-flow: row wrap;
  justify-content: flex-start;
}

main .tweet {
  background-color: #fff;
  padding: 20px;
  border-radius: 6px;
  margin: 1rem;
  flex-basis: 100%
}
```

Setting a 725px breakpoint

Now you are going to lay out these elements for a larger screen. The first thing to know is that setting a breakpoint at 725px is completely arbitrary. You just want to set the breakpoint where you feel a relaying out of the page makes sense, design-wise.

First, create a Media Query in the CSS document that will only activate if the screen is bigger than 725px:

```
/* New breakpoint around 725px */
@media only screen and (min-width: 725px) {

}
```

Remember: Put this rule at the bottom of your CSS. Since rules are applied from top to bottom, putting rules like this at the top of the `.css` file will mean that the rules inside the Media Query will be overwritten. *Media Queries always go at the bottom of the `.css` file.*

Now, you want to think about how the elements should be changed for the bigger screen. One idea would be to make the navigation elements in a row, rather than in a column. Since you have more horizontal space now, they don't need to be stacked up right on top of each other. We can do that by redefining the flex container definition:

```
/* make the nav links go to one line */
nav ul {
  flex-direction: row;
  justify-content: space-around;
}
```

Now that they are side by side and `space-around` has been defined, they look very squashed. We can tell the flex container to have each link take up a third of the width:

```
/* Each nav link should take up just a third of the width (with some
padding) */
nav ul li {
  flex-basis: 30%;
}
```

You could also make the tweets not take up the whole width, so that they can be seen two at a time. You can do that by making them take up less than half the screen:

```
/* Make it so that we have two columns of tweets */
main .tweet {
  flex-basis: 45%;
}
```

That will leave you with this:

```
/* New breakpoint around 725px */
@media only screen and (min-width: 725px) {
  /* make the nav links go to one line */
  nav ul {
    flex-direction: row;
    justify-content: space-around;
  }

  /* Each nav link should take up just a third of the width (with some
padding) */
  nav ul li {
    flex-basis: 30%;
  }

  /* Make it so that we have two columns of tweets */
  main .tweet {
    flex-basis: 45%;
  }
}
```

Setting a 900px breakpoint

Once the screen gets even bigger, you have even more real estate to work with. At 900px, the screen feels a little too roomy.

First, add your Media Query to your CSS for screens over 900px:

```
/* New breakpoint at 900px */
@media only screen and (min-width: 900px) {

}
```

Remember: Put this rule below the previous Media Query. Again, because the CSS rules are read from top to bottom, you want these rules to overwrite the ones from the 725px.

Within here, you can add rules for this even larger screen. All the rules from the 725px screen will still apply, but rules from this Media Query will override them.

First, you can start putting elements next to each other with this much extra width. Put the header and the navigation on the same horizontal line. You'll do this by redefining the grid.

```
/* Redefine the grid so that we have the nav next to the header */
body {
  grid-template-columns: 2em 1fr 1fr 2em;
  grid-template-areas:
    ". header nav ."
    ". content content ."
    ". footer footer .";
}
```

Now you have a little more gutter around the sides and two equal columns in the middle. You now have the header and the navigation side by side.

The navigation is currently stuck at the top of the header bar now. It's easy to center using Flexbox.

```
/* Center the elements in the nav to the center */
nav {
  display: flex;
  align-items: center;
  justify-content: center;
}
```

You can also make it so there are three columns of tweets.

```
/* show three tweets side by side */
main .tweet {
  flex-basis: 29%;
}
```

All of these rules together in your CSS file will look like this:

```
/* New breakpoint at 900px */
@media only screen and (min-width: 900px) {
  /* Redefine the grid so that we have the nav next to the header */
  body {
    grid-template-columns: 2em 1fr 1fr 2em;
    grid-template-areas:
        ". header nav ."
        ". content content ."
        ". footer footer .";
  }

  /* Center the elements in the nav to the center */
  nav {
    display: flex;
    align-items: center;
    justify-content: center;
  }

  /* show three tweets side by side */
  main .tweet {
    flex-basis: 29%;
  }
}
```

You can play and change these as you want to make your page look good at different sizes.