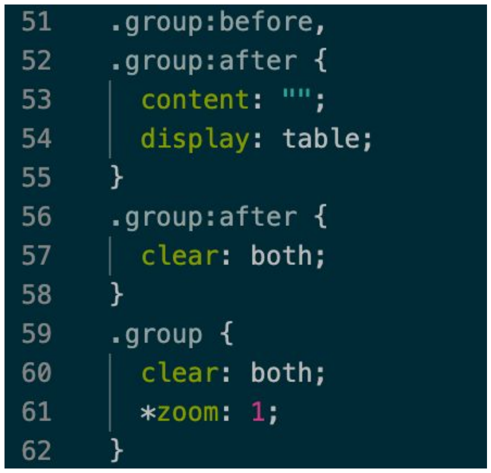
AIM Code School

Style Conference Part III

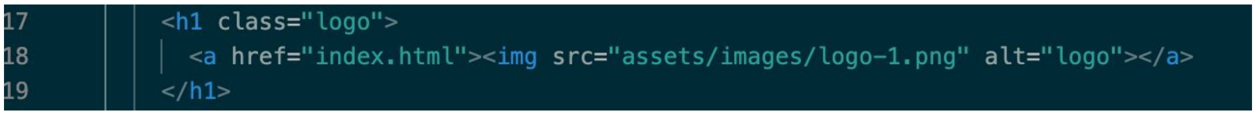
In order to correctly use the floating property, we need to provide a way to wrap those floats by adding the clearfix to our css. Inside the **main.css file**, just below the grid style, add the clearfix under the class **.group**



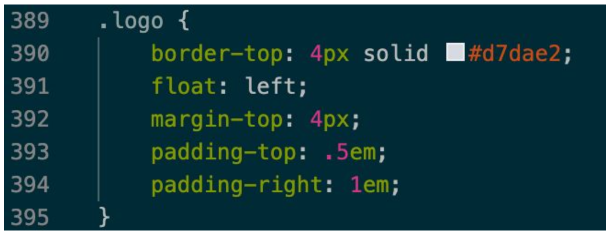
```css
.group:before,
.group:after {
  content: "";
  display: table;
}
.group:after {
  clear: both;
}
.group {
  clear: both;
  *zoom: 1;
}
```

Now that we have our containers float the primary **<h1>** within the **<header>** element to the left and allow all of the other content in the header to wrap to the right of it.

To do this, let's add a class of logo to the **<h1>** element. Then within our CSS, let's add a new section of styles for the primary header. In this section we'll select the **<h1>** element with the **logo class** and then **float it** to the **left.** And while we are at it lets go ahead and add the other css property values pairs. These pairs will consist of **paddings** and **margins** to better layout the logo **img** along with adding a **border** to the top.

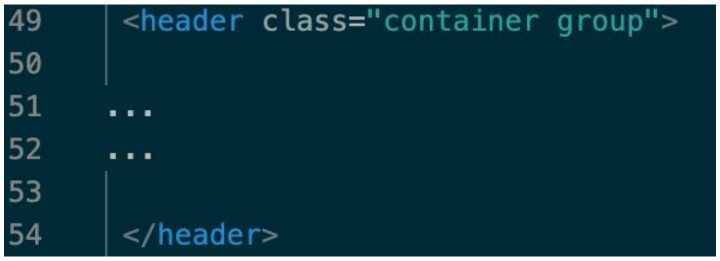


```html
<h1 class="logo">
  <a href="index.html"><img src="assets/images/logo-1.png" alt="logo"></a>
</h1>
```



```css
.logo {
    border-top: 4px solid #d7dae2;
    float: left;
    margin-top: 4px;
    padding-top: .5em;
    padding-right: 1em;
}
```

Since we floated our logo inside the <h1> element, we need to make sure to contain that float. We need to make sure it floats to the closest parent element of the <h1> element which happens to be the <header> tag. So, we need to add the class of .group to the <header> tag. We do this to apply the clearfix styles we set up earlier to the <header> element.

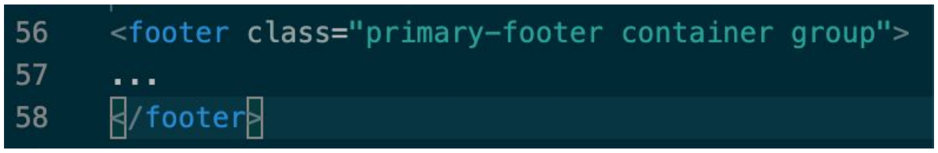


```
<header class="container group">

...
...

</header>
```

Let's now focus on the footer. Similar to what we did with the <header> element, we'll want to float our copyright information to the left within the <small> element so the other tags wrap around it to the right.

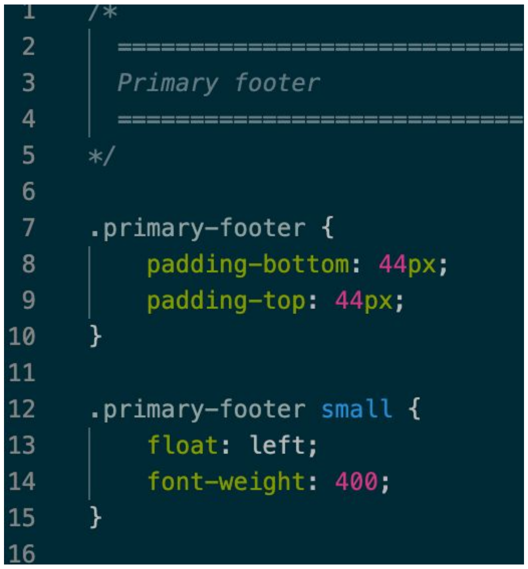
The difference is we are not going to use a class directly on the floated element. Instead, we need to apply a class to the parent of the floated element and use a unique CSS selector to select the element and then float it.

We can achieve this by adding the class of primary-footer to the <footer> tag. We know that we need to float an element within the <footer> tag, so we also need to add the class of **group** while we're at it.



```
<footer class="primary-footer container group">
...
</footer>
```

We can now use the primary-footer class to prequalify the <small> element within css. Select and float the <small> element to the left. We also want to create a new section within our main.css file for the primary footer styles.


```
/*
  ============================
  Primary footer
  ============================
*/

.primary-footer {
    padding-bottom: 44px;
    padding-top: 44px;
}

.primary-footer small {
    float: left;
    font-weight: 400;
}
```

What we are doing here is selecting the <small> element, which has to reside within an element with the class attribute value of primary-footer, such as our <footer> element, for example. Lastly, we are going to throw some padding on the top and bottom to separate it more from the rest of the page, which we will do directly in the primary-footer class.

With all of these changes to the <header> and <footer> elements, we have to be sure to make them on every page, not just the index.html page.

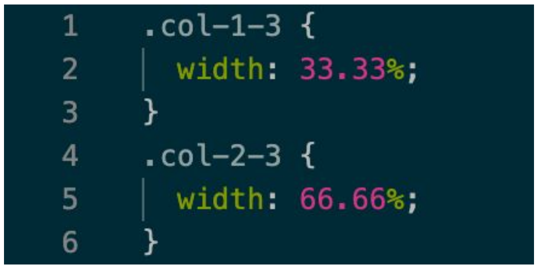
So far, this is what we have all together for our HTML....

[Click here to see the code and browser view]()

Make Sure Your Code is Correct and let's keep moving!

Now, we want to create a three-column-layout. We are going to practice coding modular style css so we can reuse the layout. We'll do this by using inline-block elements.

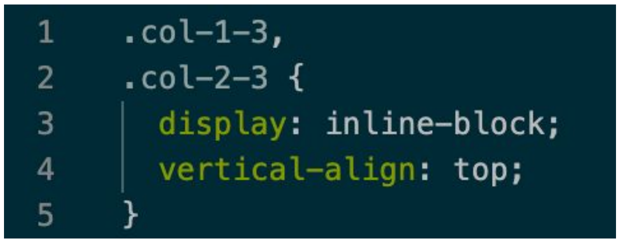
Begin by creating classes, in your **main.css** file, that define the width of these columns. We will be using the two classes **col-1-3**, for the one-third, and **col-2-3**, for two-thirds.



```css
.col-1-3 {
  width: 33.33%;
}
.col-2-3 {
  width: 66.66%;
}
```

We want both of these columns to **display as inline-block** elements and we also need to make sure we set the **vertical-alignment to top.**

Let's go ahead and do that. So make two new selectors that will combine both the display and the alignment property style.

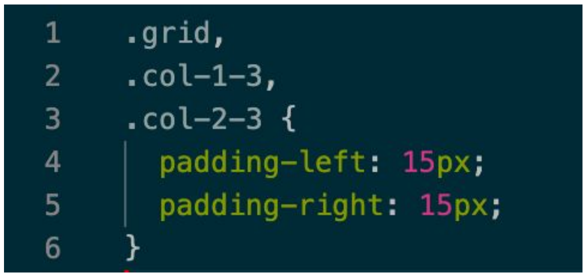


```css
.col-1-3,
.col-2-3 {
  display: inline-block;
  vertical-align: top;
}
```

Before we move on, let's talk about the syntax. Notice the two classes we created are separated by a common signifying that after the first class there will be another one to follow. The second selector is followed by a curly bracket to signify the declaration of the property:value pairs.

We want some space in between each of the columns to help break up the content. We can accomplish this by putting **horizontal padding** on each of the columns.

When two columns are next to one another, the width of the space between them will be double that of the space from the outside columns to the edge of the row. To balance this we'll place all of our columns within a **grid** and add the same **padding** from our columns to that grid.

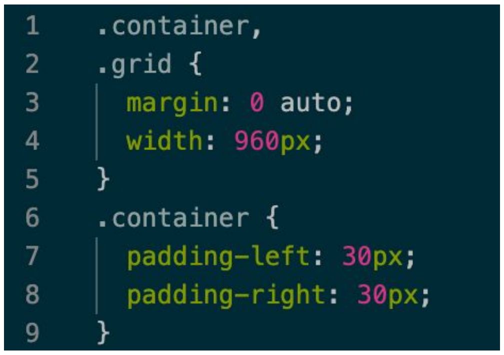


```
.grid,
.col-1-3,
.col-2-3 {
  padding-left: 15px;
  padding-right: 15px;
}
```

When we're setting up the horizontal padding, we'll need to be careful. Remember, in the last lesson we created a **container** element, known by the class of container, to center all of our content on a page within a **960-pixel-wide** element. Currently if we were to put an element with the class of grid inside an element with the class of **container**, their horizontal paddings would add to one another, and our columns would not appear proportionate to the width of the rest of the page.

We don't want this to happen, so instead, we'll have to share some of the styles from the container rule set with the grid rule set. Specifically, we'll need to share the width property and values (to make sure our page stays fixed at **960 pixels wide**) and the margin property and values (to center any element with the class of grid on the page).

We'll accomplish this by breaking up the old container rule set into the following:

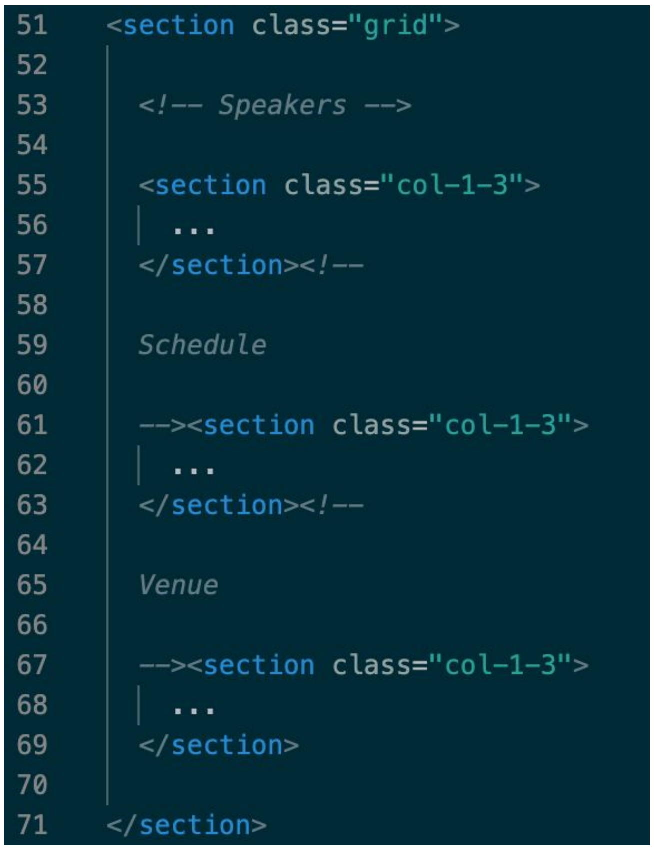


```
.container,
.grid {
  margin: 0 auto;
  width: 960px;
}
.container {
  padding-left: 30px;
  padding-right: 30px;
}
```

Now, all elements with the class of container or grid will be **960 px** wide and centered on the page. Also we've saved the existing **horizontal padding** for any element with the class of **container** by moving it into a new separate rule.

Right now the teasers on the home index.html page are wrapped in a **<section>** tag with a class of **container**. We want to change that class from **container to grid** so that we can begin to put columns within it. We also want to add the class of **col-1-3** to each of the **<section>** elements within the **<section>** tags with the class of grid.

Finally , we want to remove the empty white space between them. (Remember we are using in-line elements for our column layout). Let's do this by using **comments** with a bit of documentation noting each upcoming section. Once you've done that each **<section>** element should look similar to this.



```
<section class="grid">

  <!-- Speakers -->

  <section class="col-1-3">
    ...
  </section><!--

  Schedule

  --><section class="col-1-3">
    ...
  </section><!--

  Venue

  --><section class="col-1-3">
    ...
  </section>

</section>
```