Assignment 6: Container Queries and Custom HTML Elements

The purpose of this assignment is to get practice using container queries to implement responsive components that don't rely on media queries. You'll also get a chance to try creating a basic custom HTML element.

In the starter files, I've provided a starter project with the following:

- **container-queries.html:** this provides a basic HTML template with a JavaScript program that fetches data about Vancouver public artworks from an API and inserts it into the page. You don't have to understand much about how this script works except the HTML template from lines 36 to 45. Open this file in a browser and view it at different viewport widths to see how it works.
  DO NOT edit this file!
- **style.css**: this stylesheet sets up a basic responsive grid for the layout of the data inserted by the script in **container-queries.html.** To make it easier to see how it works, I've added some code from lines 2 to 4, but you should delete this before you get started. Besides that change,
  DO NOT edit this file!
- **custom-elements.html:** this file is similar to the other HTML file, but it inserts the public art data using a custom HTML element instead of regular HTML.
  DO NOT edit this file!
- **public-art.js:** this is the JavaScript that you will modify to implement the custom HTML element. Currently, it's just a template that will require modifications from you to work properly.
  PLEASE DO edit this file!

Also in the starter files are screenshots and a demo video to show what the site should look when complete (both with and without the custom HTML elements). Note that your site does not have to respond in exactly the same way as the viewport size changes; this is just a guide to give you an idea of what a solution could look like.

Task 1: Container Queries

Create a file called **my-styles.css;** this is where all your CSS code should be written for this task.

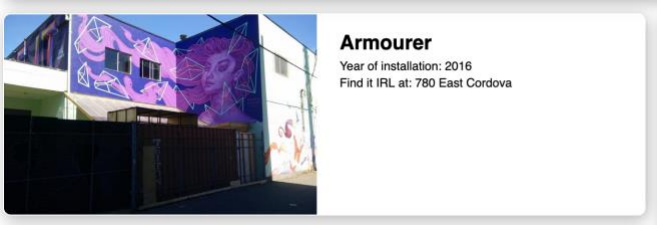
Your job is to use **container queries** to implement grid cells that respond to their size and shape depending on which area of the grid they are in and how wide the viewport is.

A few general guidelines:

- The container for the container queries should be the **div** with class **container,** on line 36 of **container-queries.html.**
- Notice that the grid from **style.css** has fixed, static heights for the rows; this means you can use **container-type: size** because both the width and height of the container will be limited by its grid cell and cannot be changed by the contents of the container.
- Images can be sized to stretch between the top and bottom of the container by using **height: 100%**.
    - To allow the image to shrink to squeeze into the grid cell, set **min-height: 100%.** (By default, the browser won't let an image get so short that it would change the aspect ratio of the image.)
    - To prevent the image from being distorted by **width** and **height** properties that don't match its intrinsic aspect ratio, use **object-fit: cover.**
- Container query breakpoints can be chosen based on what looks good to you.
- For sizing the text, use the **cqh** or **cqw** container-relative units; **1cqh = 1%** of the height of the container, and **1cqw = 1%** of the width of the container. If you want to set minimum or maximum sizes, you can use **clamp(), min(),** or **max():** **https://developer.mozilla.org/en-US/docs/Web/CSS/min**

Task 1, Part A: Small, Landscape-Oriented Containers

When a container is "small" (according to your definition of what that means) it should look something like this:

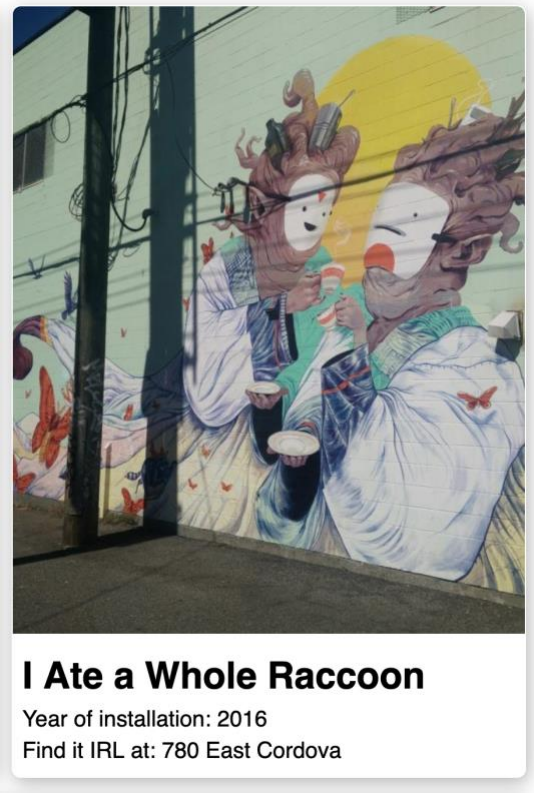


Task 1, Part B: Large, Landscape-Oriented Containers

Larger landscape-oriented containers should look more like this, with partially tansparent text container:

Task 1, Part C: Portrait Orientation

When the container has a portrait orientation, it should look like this:



You can query the orientation like this:

@container (orientation: portrait) {

}

If you need to, you can also add ***and (orientation: landscape)*** to your other queries to make sure they only apply to landscape orientation.

Task 2: Custom HTML Element

Your next task is to implement the same layout from Task 1, but using an **HTML custom element**. Look closely at lines 36 to 38 in **custom-elements.html** to see how this element will be marked up in the **light DOM**. Notice that on line 46, the script that registers the custom element, **public-art.js,** is being loaded. In **public-art.js,** I've given you a template that you can modify to implement this custom element. A few tips:

- You can *mostly* copy the HTML markup from **container-queries.html** and CSS code from **my-styles.css**, but you'll have to make a few changes based on the following:
    - Notice that the **image** is a **slotted element**, but the other information (heading text, year, and location address) are implemented as **attributes.**
    - Remember that to select a slotted element in CSS, you must use the **::slotted()** pseudo-element.

Once you've gotten the custom HTML element working, the page should look *exactly* the same as the page implemented in **container-queries.html** with **my-styles.css.**

Create a file called **my-styles--custom-elements.css**. This CSS file should do one thing and only one thing: select the **h2** headings within the custom **<public-art>** elements and add an **underline**. Remember that to select elements within the **shadow DOM** (but not slotted elements) you must place a **part** attribute on the element you'd like to select and use the **::part()** pseudo-class to select it.

Grading

- Task 1
    - [1 mark] setting up the container and using cqh and/or cqw units
    - [3 marks] one mark for each of parts A, B, and C
- Task 2
    - [1 mark] converting layout to use custom elements
    - [1 mark] adding underline to headings from light DOM stylesheet

Total: 6

As usual, up to -25% could be deducted for improper hand in, disorganized files, poorly formatted code, etc. Ask me if in doubt.

Hand In

Make sure your project folder doesn't contain any unnecessary files like instructions, the demo video, etc. Rename the folder to **a6-firstname-lastname,** zip the folder, and hand it in to Brightspace.