Responsive design

https://www.codecademy.com/courses/learn-responsive-design

With CSS, you can avoid hard coded measurements and use relative measurements instead. Relative measurements offer an advantage over hard coded measurements, as they **allow for the proportions of a website to remain intact regardless of screen size or layout**.

em represents the size of the base font being used.

Rem stands for root em. It acts similar to em, but instead of checking parent elements to size font, it checks the root element. The root element is the <html> tag.

Most browsers set the **font size of <html> to 16 pixels**, so by default rem measurements will be compared to that value. To set a different font size for the root element, you can add a CSS rule.

One advantage of using rems is that all elements are compared to the same font size value, making it easy to predict how large or small font will appear. If you are interested in sizing elements consistently across an entire website, the rem measurement is the best unit for the job. **If you're interested in sizing elements in comparison to other elements nearby, then the em unit would be better suited for the job.**

When percentages are used, elements are sized relative to the dimensions of their parent element (also known as a container). **Be careful, a child element's dimensions may be set erroneously if the dimensions of its parent element aren't set first.**

Note: Because the box model includes padding, borders, and margins, setting an element's width to 100% may cause content to overflow its parent container. While tempting, 100% should only be used when content will not have padding, border, or margin.

Although relative measurements provide consistent layouts across devices of different screen sizes, elements on a website can lose their integrity when they become too small or large. **You can limit how wide an element becomes with the following properties:**

min-width — ensures a minimum width for an element.
max-width — ensures a maximum width for an element.
You can also limit the minimum and maximum height of an element.
min-height — ensures a minimum height for an element's box.
max-height — ensures a maximum height for an element's box.

```
.container {
 width: 50%;
 height: 200px;
 overflow: hidden;
 }
 .container img {
 max-width: 100%;
 height: auto;
 display: block;
 }
```

In the example above, `.container` represents a container div. It is set to a width of `50%` (half of the browser's width, in this example) and a height of 200 pixels. Setting `overflow` to `hidden` ensures that any content with dimensions larger than the container will be hidden from view.

The second CSS rule ensures that images scale with the width of the container. The `height` property is set to `auto`, meaning an image's height will *automatically* scale proportionally with the width. Finally, the last

line will display images as block level elements (rather than inline-block, their default state). This will prevent images from attempting to align with other content on the page (like text), which can add unintended margin to the images.

It's worth memorizing the entire example above. It represents a *very common* design pattern used to scale images and videos proportionally.

Note: The example above scales the width of an image (or video) to the width of a container. If the image is larger than the container, the vertical portion of the image will overflow and will not display. To swap this behavior, you can set `max-height` to `100%` and `width` to `auto` (essentially swapping the values). This will scale the *height* of the image with the height of the container instead. If the image is larger than the container, the horizontal portion of the image will overflow and not display.

Background images of HTML elements can also be scaled responsively using CSS properties.

```
body {
  background-image: url('#');
  background-repeat: no-repeat;
  background-position: center;
  background-size: cover;
  }
```

In the example above, the first CSS declaration sets the background image (`#` is a placeholder for an image URL in this example). The second declaration instructs the CSS compiler to not repeat the image (by default, images will repeat). The third declaration centers the image within the element.

The final declaration, however, is the focus of the example above. It's what scales the background image. The image will *cover* the entire background of the element, all while keeping the image in proportion. If the dimensions of the image exceed the dimensions of the container then only a portion of the image will display.

When a website responds to the size of the screen it's viewed on, it’s called a **responsive website**. Because websites can be displayed on thousands of different screen sizes, they must be able to respond to a change in screen size and adapt the content so that users can access it.

CSS uses ***media queries*** to adapt a website's content to different screen sizes. With media queries, CSS can detect the size of the current screen and apply different CSS styles depending on the width of the screen.

```
@media only screen and (max-width: 480px) {
  body { font-size: 12px;
  }
}
```

The example above demonstrates how a media query is applied. The media query defines a rule for screens smaller than 480 pixels (approximately the width of many smartphones in landscape orientation).

Let’s break this example down into its parts:

1. `@media` — This keyword begins a media query rule and instructs the CSS compiler on how to parse the rest of the rule.
2. `only screen` — Indicates what types of devices should use this rule. In early attempts to target different devices, CSS incorporated different media types (screen, print, handheld). The rationale was that by knowing the media type, the proper CSS rules could be applied. However, “handheld” and “screen” devices began to occupy a much wider range of sizes and having only one CSS rule per media device was not sufficient. `screen` is the media type always used for displaying content, no matter the type of device. The `only` keyword is added to indicate that this rule only applies to one media type (`screen`).

3. and (max-width : 480px) — This part of the rule is called a *media feature*, and instructs the CSS compiler to apply the CSS styles to devices with a width of 480 pixels or smaller. Media features are the conditions that must be met in order to render the CSS within a media query.
4. CSS rules are nested inside of the media query's curly braces. The rules will be applied when the media query is met. In the example above, the text in the bodyelement is set to a font-size of 12px when the user's screen is less than 480px.

Range

Specific screen sizes can be targeted by setting multiple width and height media features. min-width and min-height are used to set the minimum width and minimum height, respectively. Conversely, max-width and max-height set the maximum width and maximum height, respectively.

By using multiple widths and heights, a range can be set for a media query.

```
@media only screen and (min-width: 320px) and (max-width: 480px) {
  /* ruleset for 320px - 480px */ }
```

The example above would apply its CSS rules only when the screen size is between 320 pixels and 480 pixels. Notice the use of a second and keyword after the min-width media feature. This allows us to chain two requirements together.

The example above can be written using two separate rules as well:

```
@media only screen and (min-width: 320px) {
  /* ruleset for 320px - 479px */
 }
@media only screen and (min-width: 480px) {
  /* ruleset for > 480px */
 }
```

The first media query in the example above will apply CSS rules when the size of the screen meets or exceeds 320 pixels. The second media query will apply CSS rules when the size of the screen meets or exceeds 480 pixels, meaning that it will override the CSS rules present in the first media query.

Both examples above are valid, and it is likely that you will see both patterns used when reading another developer's code.

Dots Per Inch (DPI)

Another media feature we can target is screen resolution. **Many times we will want to supply higher quality media (images, video, etc.) only to users with screens that can support high resolution media.** Targeting screen resolution also helps users avoid downloading high resolution (large file size) images that their screen may not be able to properly display.

To target by resolution, we can use the min-resolution and max-resolution media features. These media features accept a resolution value in either dots per inch (dpi) or dots per centimeter (dpc). Learn more about resolution measurements here.

```
@media only screen and (min-resolution: 300dpi) {
  /* CSS for high resolution screens */
 }
```

The media query in the example above targets high resolution screens by making sure the screen resolution is at least 300 dots per inch. If the screen resolution query is met, then we can use CSS to display high resolution images and other media.

We can mix the two as well:

```
@media only screen and (max-width: 480px) and (min-resolution: 300dpi) {

  /* CSS ruleset */ }Comma Separated List
```

If only one of multiple media features in a media query must be met, media features can be separated in a comma separated list.

For example, if we needed to apply a style when only one of the below is true:

- The screen is more than 480 pixels wide
- The screen is in landscape mode

We could write:

```
@media only screen and (min-width: 480px), (orientation: landscape) { /* CSS ruleset */ }
```

In the example above, **we used a comma (,) to separate multiple rules. The example above requires only one of the media features to be true for its CSS to apply.**

Note that the second media feature is orientation. The orientation media feature detects if the page has more width than height. If a page is wider, it's considered landscape, and if a page is taller, it's considered portrait.

We know how to use media queries to apply CSS rules based on screen size and resolution, but how do we determine what queries to set?

The points at which media queries are set are called *breakpoints*. Breakpoints are the screen sizes at which your web page does not appear properly. For example, if we want to target tablets that are in landscape orientation, we can create the following breakpoint:

```
@media only screen and (min-width: 768px) and (max-width: 1024px) and (orientation: landscape) { /* CSS ruleset */ }
```

The example above creates a screen size range the size of a tablet in landscape mode and also identifies the orientation.

However, setting breakpoints for every device imaginable would be incredibly difficult because there are many devices of differing shapes and sizes. In addition, new devices are released with new screen sizes every year.

Rather than set breakpoints based on specific devices, the best practice is to resize your browser to view where the website naturally breaks based on its content. **The dimensions at which the layout breaks or looks odd become your media query breakpoints.** Within those breakpoints, we can adjust the CSS to make the page resize and reorganize.

By observing the dimensions at which a website naturally breaks, you can set media query breakpoints that create the best possible user experience on a project by project basis, rather than forcing every project to fit a certain screen size. Different projects have different needs, and creating a responsive design should be no different.

Review: Media Queries

- When a website responds to the size of the screen it's viewed on, it’s called a responsive website.
- You can write media queries to help with different screen sizes.
- Media queries require media features. Media features are the conditions that must be met to render the CSS within a media query.
- Media features can detect many aspects of a user's browser, including the screen's width, height, resolution, orientation, and more.
- The and operator requires multiple media features to be true at once.
- A comma separated list of media features only requires one media feature to be true for the code within to be applied.
- The best practice for identifying where media queries should be set is by resizing the browser to determine where the content naturally breaks. Natural breakpoints are found by resizing the browser.

Review: Relative Measurements

- Content on a website can be sized relative to other elements on the page using relative measurements.
- The unit of em sizes font relative to the font size of a parent element.
- The unit of rem sizes font relative to the font size of a root element. That root element is the <html> element.
- Percentages are commonly used to size box-model features, like the width, height, borders, padding, or margin of an element.
- When percentages are used to size width and height, child elements will be sized relative to the dimensions of their parent (remember that parent dimensions must first be set).
- Percentages can be used to set padding and margin. Horizontal and vertical padding and margin are set relative to the width of a parent element.
- The minimum and maximum width of elements can be set using min-width and max-width.
- The minimum and maximum height of elements can be set using min-height and max-height.
- When the height of an image or video is set, then its width can be set to auto so that the media scales proportionally. Reversing these two properties and values will also achieve the same result.
- A background image of an HTML element will scale proportionally when its background-size property is set to cover.

Relative units of measurement are a first step towards incorporating responsive design in a website. When combined with more advanced responsive techniques, you can create a seamless user experience regardless of a device's screen size.