{ Responsive Units. }

Objectives:

By the end of this chapter, you should be able to:

- List different units of measurement for responsive design
- Compare and contrast `em` and `rem` units
- List some tips for writing cleaner HTML and CSS

Different units of measurement

When it comes to setting widths, font sizes, or other things require some measurement of length, we've seen two different units of measurement: pixels and percentages. Pixels are an *absolute unit* of measurement. This means that they are great when you know "exactly" what the dimensions for the property you're setting should be. Pixels, inches, centimeters, millimeters, and points are all examples of absolute units of measurement.

In contrast, *relative units*, like the name suggests, are always relative to the length of another property. Examples here include percentages, ems, root ems, exes, character units, viewport width, viewport height, viewport minimum, and viewport maximum.

- `%` - As we've seen, this sets the length to be a percentage of whatever container or fixed value the element is inside of.
- `em` - `1em` is equal to the default `font-size` for the element. This lets you set properties involving length relative to this font size. Note that if you define the `font-size` in terms of `em`s, `1em` will correspond to the `font-size` of the parent.
- `rem` - just like em's but they are only relative to the root (i.e. `html`) element. This means that `1rem` corresponds to the same size everywhere on the page.
- `vh` - `1vh` is equal to 1/100th of the viewport's height.
- `vw` - `1vw` is equal to 1/100th of the viewport's width.
- `vmin` - `1vmin` is equal to the minimum of `1vh` and `1vw`.
- `vmax` - `1vmax` is equal to the maximum of `1vh` and `1vw`.

Writing cleaner HTML + CSS

Here are three general tips for helping write cleaner and more maintainable HTML and CSS:

1. For smaller projects, shy away from using IDs and lots of classes and instead use more complex selectors (descendant, adjacent, sibling, n-th child, etc).

2. Keep your CSS DRY by adding common styles to selectors separated by a comma.

```
/* BAD */

.heading_content {
    color: #0000F9;
    font-size: 2em;
    margin-top: 1em;
    text-align: center;
}

.footer_content {
    color: #E046CF;
    font-size: 2em;
    margin-top: 1em;
    text-align: center;
}

/* BETTER */

.heading_content, .footer_content {
    font-size: 2em;
    margin-top: 1em;
    text-align: center;
}

.heading_content {
    color: #0000F9;
}

.footer_content {
    color: #E046CF;
}
```

3. As you start working with larger CSS files, there are many different patterns for structuring your CSS. You can get started by reading and learning about CSS organizational structures like BEM, which stands for **B**lock, **E**lement, and **M**odifier, is a way of structuring CSS classes to help write more maintainable CSS as your project grows. For more on BEM, check out this article.