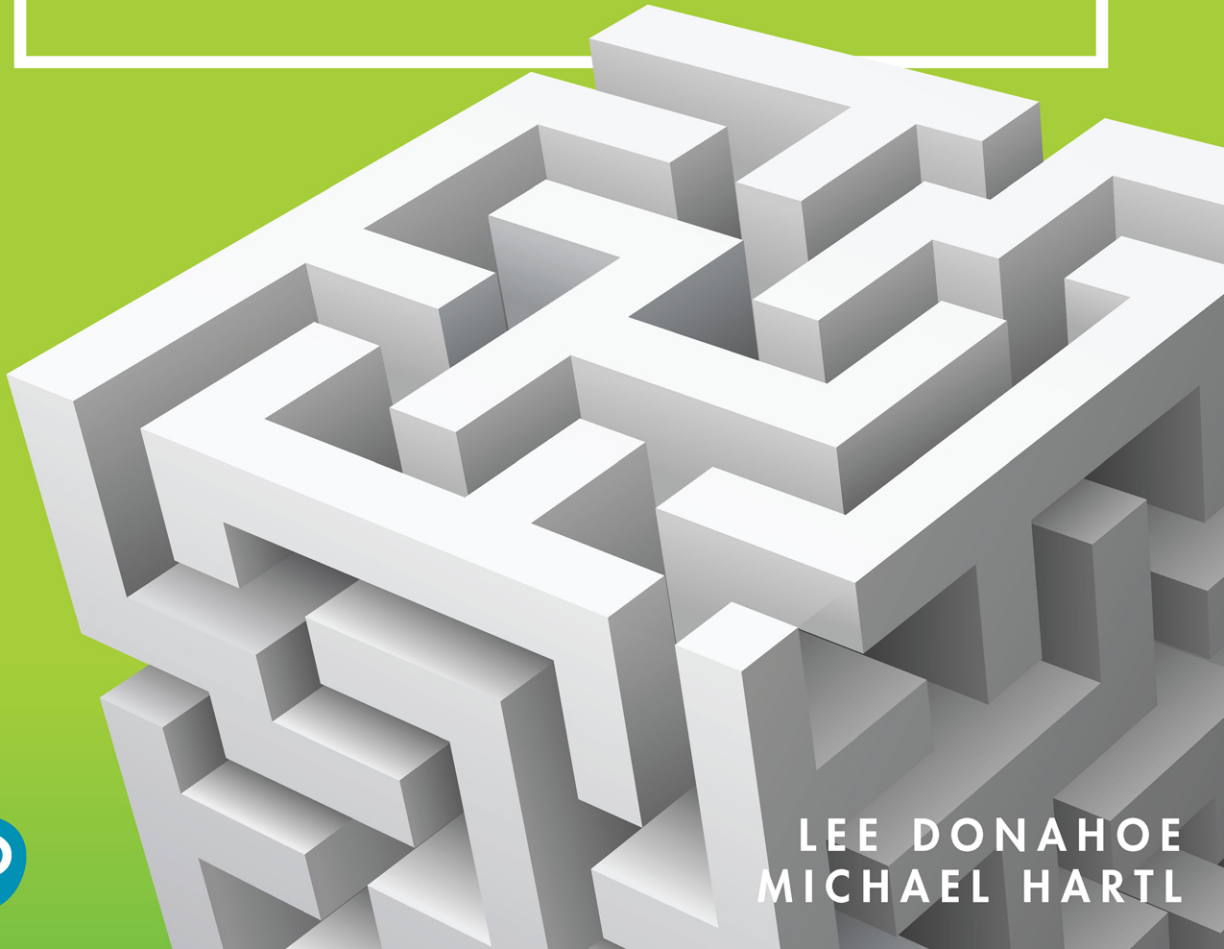
learn enough

# HTML, CSS AND LAYOUT TO BE DANGEROUS



LEE DONAHOE
MICHAEL HARTL

# Praise for Learn Enough Tutorials

"I have nothing but fantastic things to say about @LearnEnough courses. I am just about finished with the #javascript course. I must say, the videos are mandatory because @mhartl will play the novice, and share in the joy of having something you wrote actually work!"

—Claudia Vizena

"I must say, this Learn Enough series is a masterpiece of education. Thank you for this incredible work!"

—Michael King

"I want to thank you for the amazing job you have done with the tutorials. They are likely the best tutorials I have ever read."

—Pedro Iatzky

`em` it will size the object based on the font size inside that element. For example, if the calculated font size in an element ends up being `16px`, and you set the padding to `1.5em`, then the padding will end up getting set to $1.5 \times 16 = 24$`px`.

Sound confusing? Figure 7.18 shows a quick diagram to explain.

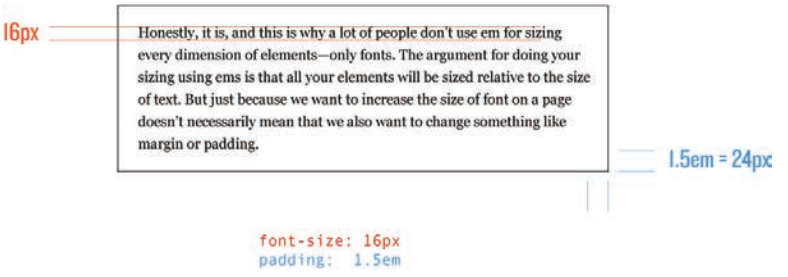



**Figure 7.18:** Dimensions inside the box get calculated based on the font size.

The argument for doing your sizing using ems is that all your elements, and their attributes like padding or margin, will be sized relative to the size of the text. But just because we want to increase the size of fonts on a page doesn't necessarily mean that we also want to change something like margin or padding. Sometimes you want the boxes that hold the content to stay the same and only the stuff inside to change, so styling every dimension of a container based on the text it contains can be inconvenient. As a result, in this tutorial we'll use the `em` unit primarily for fonts, but will also use it in margins or padding where it is helpful to have the sizing be responsive.

*Warning*: As is often the case with such subjective judgments, this is holy-war territory (Figure 7.19).[3]

3. Image courtesy of melnyk58/123RF.


**Figure 7.19:** Well… that's just, like, your opinion, man.

### 7.5.1 Exercises

1. Give the `.bio-box` class a padding of `2.5em`, and see how at small sizes the padding seems reasonable.
2. Set the `.bio-box` font size to `48px`, and see how now your boxes' padding is a significant percentage of the entire browser width.
3. Set the `.bio-box` padding to a pixel size of `20px` to see how that can make the space sizing independent from the content sizing.

## 7.6 `rem` Isn't Just for Dreaming

The cumulative effect of the `em` unit (Section 7.5) can at times make designing layouts difficult since it makes it harder to drop sections of a page into other sections and be confident that you aren't going to end up with some weird cumulative sizing issue. (Recall the goal from Section 6.4 to make our markup as modular and LEGO-like as possible.) In the years since the original release of CSS, browsers have implemented a new relative unit that allows for us to create modular sections that can be placed on the page without sizing uncertainties, the *root em*, or `rem`.

This `rem` unit works similarly to `em`, in that it is a percentage of an absolute font size, but instead of being cumulatively sized based on the whole parent–child tree, the `rem` unit *always* refers back to the font size of the `html` tag—in other words, it always refers to the most basic font size for the whole page. (As noted in Section 7.5, this default size is `16px`.)

In effect, the `rem` unit works like a document-wide setting, so you can set the size of elements like boxes, or font sizes, and have them all tie back to a single value: the font size of the `html` element. If you want to make everything a little bigger, or smaller, you can change just this one font size and everything on the page adapts in a controlled manner.

`rem` is especially useful in combination with `em` units in developing modules. The best practice is to set a font size for the module's wrapper using a `rem` unit, and then style the fonts inside using `em` units. Because `rem` values are absolute (in relation to the page font size), you don't need to worry that the cumulative nature of `em` will keep going up the parent tree and make everything in the box tiny or huge (Section 7.5). This kind of styling allows you to create modules that can be safely dropped into any part of a page, while keeping the advantages of using relative font sizes.

To see this in action, let's set the size of the `.bio-box` to `1rem` and then add in a new declaration to set the header `h3`s to be `1.5em` and the `.bio-copy` to be `1em`.

Listing 7.15 has the entire CSS block at this point—copy and paste if you aren't synced up.[4] (Note that Listing 7.15 removes the `h2` rule from Listing 7.5.)

**Listing 7.15:** The CSS section up to now, with new font sizes.
*index.html*

```
<style>
  /* GLOBAL STYLES */
  a {
    color: #f00;
  }

  /* SOCIAL STYLES */
  .social-link {
    background: rgba(150, 150, 150, 0.5);
    color: blue;
  }

  /* BIO STYLES */
```

4. Recall that the code listings are available at https://github.com/learnenough/learn_enough_html_css_and_layout_code_listings.

```
  .bio-wrapper {
    font-size: 24px;
  }
  .bio-box {
    border: 1px solid black;
    font-size: 1rem;
    width: 50%;
  }
  .bio-box h3 {
    font-size: 1.5em;
  }
  .bio-copy {
    font-size: 1em;
  }
  .bio-copy a {
    color: green;
  }
</style>
```

---

Now the whole page will be set to be the same size as the `html` default font size of `16px`, and the header in the bio box will always be one and a half times that size even though the `.bio-wrapper` is set to a very large `24px`. Meanwhile, the bio copy will remain at the default size for the page. If we decide that all the copy on the site should be bigger, we can easily reset the default size with a rule like

```
html {
  font-size: 18px;
}
```

With the change in Listing 7.15, all the copy in the boxes will resize but will stay in proportion without any cumulative effects (Figure 7.20). All of those benefits become important the moment you need to design a site that looks good on different devices, such as a desktop computer and a mobile phone. We'll discuss this important issue further in Chapter 13. (If you added the 18px font size styling to the `html` element, go ahead and delete it now.)

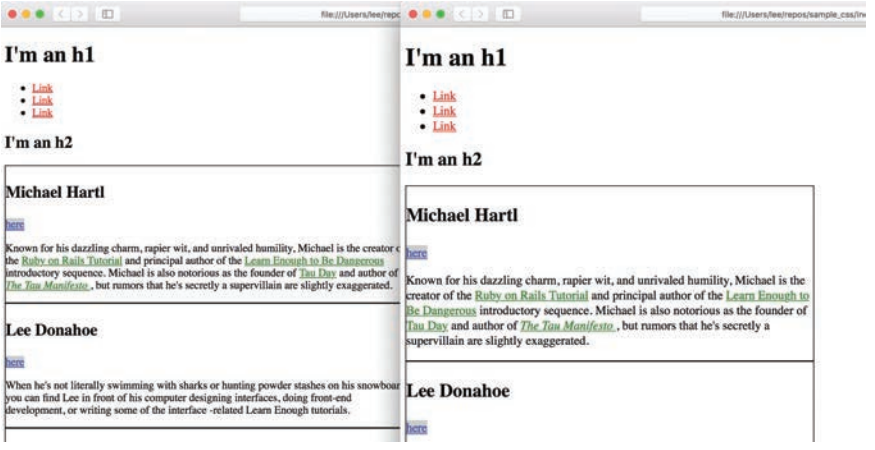



**Figure 7.20:** The 16px base font size is on the left; the 18px size is on the right, with everything scaled proportionally.

### 7.6.1 Exercises

1. Copy the entire first `.bio-box` and paste it inside the `h1`. You should see that `rem` sizing allowed for the whole section to be modular and retain the set styling.
2. In the CSS, change the font size for `.bio-box` from `1rem` to `1em` and notice the effect.

## 7.7 vh, vw: The New(er) Kids on the Block

Speaking of mobile-friendly units, we arrive now at two newer dimensional units that are also incredibly useful for responsive (mobile) layouts: the viewport height, `vh`, and viewport width, `vw`. These units allow us to size elements on the page based on the actual size of the browser window or mobile device screen. Each `vh` or `vw` is 1% of the corresponding screen dimension, so `3vh` would equal 3% of the height of the screen and `100vw` would be 100% of the width.

Neither `vh` nor `vw` is affected by parent elements, and neither has any weird cumulative inheritance issues—everything is determined by the size of the browser window or device screen. Up until relatively recently these units weren't reliably supported by all browers, but as long as a good percentage of your users aren't using really old

browsers you can safely use **vh** and **vw** to do some fun things, like design sections that fill the browser window no matter what the size of that window is.

We'll apply the viewport units as part of adding a *hero section* to our site, a design pattern that involves having an attention-grabbing area at the top of the page containing a dramatic image, a call to action, etc. We'll start by wrapping the top section of our test page in a new **div** with two classes, **.full-hero** and **.hero-home** (Listing 7.16). (We'll use **.full-hero** starting in Section 8.3 and **.hero-home** starting in Chapter 10.)

**Listing 7.16:** Adding a wrapper around the content, and giving it class names.
*index.html*

```
<div class="full-hero hero-home">
  <h1>I'm an h1</h1>
  <ul>
    <li>
      <a href="https://example.com/" class="social-link">Link</a>
    </li>
    <li>
      <a href="https://example.com/" class="social-link">Link</a>
    </li>
    <li>
      <a href="https://example.com/" class="social-link">Link</a>
    </li>
  </ul>
</div>
```

Note that Listing 7.16 also includes the results of the exercise in Section 7.1.3 that added the **.social-link** class to the link inside each **li**.

With the classes defined in Listing 7.16, we're in a position to start giving the hero section some styles. We'll start by adding a background color and a height equal to 50% of the viewport using **50vh**, as shown in Listing 7.17.

**Listing 7.17:** Adding a height based on browser size.
*index.html*

```
/* HERO STYLES */
.full-hero {
  background-color: #c7dbfc;
  height: 50vh;
}
```