Typography

Daniel Quinn

So, what we like to talk about in this tutorial are different ways of applying fonts to websites. And that's custom fonts outside of the web-safe ones that come with a browser.

There are three methods of applying fonts. That includes web-safe fonts, which are the ones that come with browsers and can be expected to work across all different users' machines, whether they're, you know, Windows machines or Macs. There is applying an embedded font from a different website. So, that would be like Google Fonts or Typekit. And then there's a method of hosting a font that's embedded on your server and then sending that to the visitor. And so, we will be using a custom font for that from DaFont.com.

In the other half of this tutorial, what we'd like to talk about is handling font sizes for those fonts once you've applied them. That includes different unit sizes, from pixels to ems to rems to view widths, which are all different methods of handling sizing for a font, some of which are absolute and don't scale with the browser, and some of which are flexible and scale with the content that they're inside of.

The fonts you're most used to are web-safe fonts. So, you'll see here I've just have some markup, which includes a class to using Tahoma and then just some text. Some of the text is bolded here, and then some of the text is italicized. And you'll see the effects of bolding and italicizing. And these are just the default styles that come in the browser.

When we look at the style sheet for this, specifically the Tahoma class, all we're doing is saying font family, we're specifying Tahoma first, and then we're specifying a list of failsafe fonts essentially. So, if Tahoma isn't available on the system, we're going to say Helvetica. We want Helvetica next. And then we want Arial, and then any old Sans-serif will suffice.

The reason why we specify Helvetica first is because Macs and Windows both have Arial. So, if we specified that first, it would always be Arial. So, the order in which we specify them matters because it's going to go for them one after the other. You can--what web-safe means is we have a pretty high certainty that the font will be available on the user system. So, we are giving some leeway in terms of how we want it to be presented.

However, let's say we want it to have a very specific font. So, we want Roboto, which is a cool Google Font. To get a Google Font, all you have to do is go to Google Fonts. And essentially, what you're doing is linking to it. So, we're going to search for Roboto. And I think the one we picked in our tutorial is--well, it's just Roboto. Is it Roboto, or is it Roboto Slab? Let me check. Oh, it's Roboto Slab in particular. So, I'm going to go and pick Roboto Slab. So, I'm going to add it to my kit and then open my kit up.

And then, you'll see that it gives you a style sheet to use. So, we want to customize that style sheet because, at the moment, it's just giving us Roboto Slab's default weight, which is regular. So, we're going to customize it. And we actually want all of them in this case. I wouldn't recommend doing that in production because, as you can see, the load time--it's not indicating it here, but you're gonna have a lot more weight in terms of the file if you add all the weights from any particular web font.

So, I'm going to choose the embed, and you'll notice that it added in the whole font a bunch of other sizes to choose from. These are different thicknesses to the font. So, I'm going to copy that. It also tells you how to reference the font.

So, if I go back and look at my tutorial document, you'll see I made a new class called Roboto, which is what's applied to each of these here. And I've specified I want Roboto Slab first, and then we'll fall back to Helvetica, then Arial, then any Sans-serif.

I've also decided to show each of the different weights that you can apply. So, a thin weight is handled by just the number 100. And so those weights go from 100 to 900, with 900 being a heavy black variation of the font. And then, of course, you know, you've got in the middle about 300. Not every font will have all the different weights, but it lets you have some flexibility in terms of, you know, the style of the font.

Finally, the last method we want to use is a custom font that may not be available on Google Fonts. So, this is a font from DaFont.com, which tends to have fonts that are either freely licensed or commercially licensed that you can choose from. And when you pick one out, you'll be able to download the font file. What you can then do is--it'll give you usually a TTF or an OTF file, which is sufficient for using on an operating system. But for the web, we need to have a WAF file.

So, the way we can have that generated is if, for one, the licensing allows. So, if you've paid for a commercial font or the font is free to use, you can then take it to a website like Font Squirrel. So, Font Squirrel. And what Font Squirrel will do is take the files that you downloaded and then convert them into--so, if we go to the generator--into the WAF file we need. So, we would just, you know, choose optimal settings. We make sure we're, you know, choosing a legal font to do this with. We upload it, and it'll spit out a WAF file.

So, in our tutorial, we have a font that is free to use called Capture It. And you'll see that the two files are in my directory here with everything else. And so, similar to how we attached a style sheet to a chat to include the Google Font, we're attaching a reference to the WAF files for this custom font, but we're doing it inside of the style sheet.

So, if you go to the style sheet that's attached--and you'll notice that the style sheet where I'm doing everything is also attached to this document--if I go inside the style sheet, you'll see at the very bottom here, I have a special rule that's saying I want to include this font face and I'm referencing my files here. So, there's the WAF, too, in the WAF files.

Most modern browsers now really just need WAF. In the past, there was EOT for Internet Explorer, and TTF was referenced. But now we just need WAF. And again, we've made a name for the font, which is Capture It. And then, we have our class applied to the Capture It markup. So, if you look back on the markup here, we have our P class, which has Capture It. So, it's just like any other font. We just have to specify the font family, and then it'll apply the custom font. What this does is it downloads the file onto the user's--visitor's local machine. That way, they're able to apply the font.

So, those are the different ways of applying custom fonts to a website. You can do web-safe, which guarantees that it will work on any old browser, any web visitor's machine. You can do Google Fonts by attaching a spreadsheet--or a style sheet from a third party. Typekit is another provider like Google Fonts. Or you can download a particular file, host it on your server, and then that'll be served to the visitor when they visit the page.

So, the next question is, well, how do we size our fonts once we have these custom fonts in place? Traditionally, there's a whole bunch of different ways to do that. The most basic way to do that is to use pixels. Pixels are an absolute sizing unit. And most browsers default to like about 16 pixels. So, I've specified specifically that I want all my paragraphs here to be 16 pixels by default.

So, then when I apply-- if we look at the markup here--my first class, which is just using a class pixel, just so I can know that it's pixels. I want to start with 22 pixels. So, if we look at the pixel class, I've said this is font size 22. So, it's just overriding the paragraph, and it's applying font size 22 pixels.

Now, one of the other options we have is ems. And what ems let us do is--the font size of an em is proportional to the element it's inside of. So, when we start with an em, the font size that it's basing itself on is what's in the body. So, the font size is a ratio based on 16 pixels. So, if we look at the em class here, I'm trying to get to 22 with the equivalent of 22 pixels. So, I have to specify in ems, which is 1.375 ems. And that's based on the fact that my body or the element that's above this one relative to it is the paragraph itself at 16.

What's useful about ems is that if you're sizing elements in ems, the font itself will scale up and down, depending on the dimensions of what it's contained inside of. But the disadvantage to ems is that they inherit each other. So, if you look at this next line item, if I--I have a nested element, so if I go back to my markup here, I've got my ems and pixels here, but then I have a nested element. So, the same class, but there's another paragraph inside of that.

So, instead of it basing its size on, you know, the body, now it's basing its size on the element it's inside of. And this element is saying it's 18 pixels. So, it becomes a little bit unpredictable as you start to nest elements inside each other, the exact font size you're working with.

Another unit size that's similar to ems but gets around that is rems. So, rems will always base themselves on the body's font size rather than a relative size, but they have the advantage of being scalable like regular ems are. So, it's the same sizing method, where it's a ratio based on that number.

The last method is a more modern way of sizing fonts called view widths. And with view widths--I may have said view heights earlier, but what I meant was view widths. View widths, the size of the font is based on the viewport, which is the VW, the viewport of the browser. So, as your browser scales up and down, the font size itself will change.

So, here, we're trying to specify a point to size against, which is 16 pixels, and a percentage of the browser's viewport. Because we don't really want it to be a full viewport width because we can't quite get the 22 pixels, which is what we want. So, what we're saying here is we want to start with, you know, a size to base it off of, which is 16 pixels, and then make it a percentage of the view width, which is about 75%. This will allow this font to scale up and down, depending on the browser's size--the browser window's size.

The one issue with view widths is we don't want it to continually scale down to infinity or scale up to infinity. So, we might have to specify through other methods minimum and maximum font sizes. That way, we can kind of control how it scales. One way to do that when we're writing modern CSS is to do it in SAS. We have some more functions to work with, which is just not available in CSS.

So, we can have a function that might rely on specifying things, first in pixels and then, for other browsers that can support it, view widths. So, we have some flexibility and SAS to handle that. But for our purposes, pixels might be sufficient because pixels nowadays can scale up and down, even if the user scales up their browser.

So, the kind of use and ubiquity of rems and ems has kind of fallen to the wayside. I would say, nowadays, you can get away with a combination of view widths and pixels to make sure that your text is scalable, while at the same time supported across the majority of browsers.