



GYMNASIUM

---

# RESPONSIVE WEB DESIGN FUNDAMENTALS

---

*Lesson 3 Handout*

*Typography & Layout*

# ABOUT THIS HANDOUT

This handout includes the following:

- A list of the core concepts covered in this lesson
- The assignment(s) for this lesson
- A list of readings and resources for this lesson including books, articles, and websites mentioned in the videos by the instructor, plus bonus readings and resources hand-picked by the instructor
- A transcript of the lecture videos for this lesson

## CORE CONCEPTS

1. The ability to reliably embed web fonts has only been available since 2009, with the widespread adoption of the CSS property `@font-face`. Prior to this, there were a number of inadequate options such as using an image or limiting the font choices to a limited set of “web-safe” fonts.
2. Although web fonts are extremely useful, it is important to remember that font files are assets that need to load, and the page won’t render until they do. This can create a number of problems ranging from a blank page to the F-O-U-T or “Flash Of Unstyled Text” which is what happens when the default typeface appears for a moment and then is replaced with the web fonts. This visual behavior can be mildly annoying to the reader, but in some cases could disrupt the reading process.
3. To avoid the flash of unstyled or invisible text, Google Web Fonts uses a JavaScript-based solution called the Web Font Loader which helps render the page immediately. In this course, we use a similar JavaScript-based solution called [Font Face Observer](#), which ties into a relatively new addition to CSS called the [Font Loading API](#).
4. Artificially throttling your connection in order to simulate a slow network is a good way to test the behavior and styling of your page in general and web fonts in particular. Most modern web browsers have this capability within the browser developer tools.
5. One strategy to improve the user experience of your website is to make sure your default typography styles (which include aspects such as typeface and line-height) match your primary web-font styles as close as possible.
6. Setting your body style to 100% or 1em, is a best practice when it comes to defining the foundation of your type hierarchy, as this size of type is universally readable across all devices.
7. Defining the style of your paragraphs is an extremely important step in building the rest of your page. In addition to the typeface itself, other aspects such as line-height and letter-spacing should be tested with real content. Once you are satisfied with your paragraph styles, move on to other components such as headings.

# ASSIGNMENTS

- Quiz
- For the second assignment, you're going to explore font pairing with [Google Fonts](#).. To begin with: open up the file named **"resume\_assignment.html"** located in the Lesson 3 folder. This is a stripped down version of the resume page used in our portfolio site. Additionally, this file is referencing a CSS document named **styles\_assignment.css**, so there is no danger of changing the original style sheet.
  - Begin exploring different typography choices for this page, it will help if you replace the content here with your own resume information, ultimately this page could become the foundation of your own responsive portfolio. However If you're not ready or comfortable with that step, feel free to leave the content as is.
  - Next, try experimenting with some different font pairings for your headings and body text and see how they change the look and feel of the content. If you want to speed things up, the Google Font site has a couple of articles in their ["featured" section](#) that suggest a number of font pairings for you, or browse through the site and try your own combinations. You will need to replace the link on line 42 with your new font choices. Then you will obviously need to go into the styles\_assignment CSS file and update the various styles with your new font choices.
  - Once you have the styles working correctly, ask yourself what changes do the new typefaces suggest? Do you need to adjust the size and or the line height? What about other styles such as bolding and Italic?
  - Once you have something you're happy with, share a link to your assignment on the classroom forum and take a few moments to also comment on one of your classmate's assignment as well.

# INTRODUCTION

*(Note: This is an edited transcript of the Responsive Web Design Fundamentals lecture videos. Some students work better with written material than by watching videos alone, so we're offering this to you as an optional, helpful resource. Some elements of the instruction, like live coding, can't be recreated in a document like this one.)*

We're going to look at a brief history of type on the web and figure out how we got here. We're going to dive into responsive typography and learn how to implement and use web fonts well. And then we're going to talk about layout, CSS columns and Flexbox, which we'll bring into our current project, and then talk about the future of layout, getting beyond just having a bunch of squishy boxes.

## CHAPTER 1: A BRIEF HISTORY OF TYPE ON THE WEB

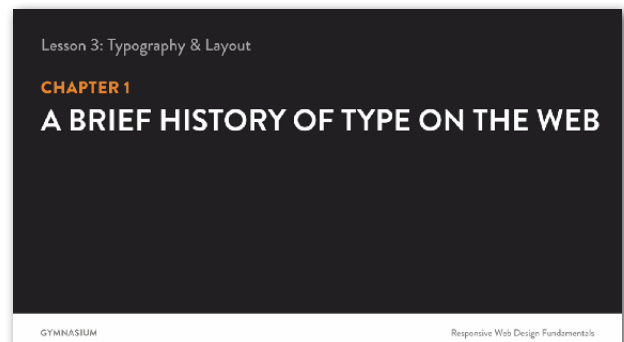
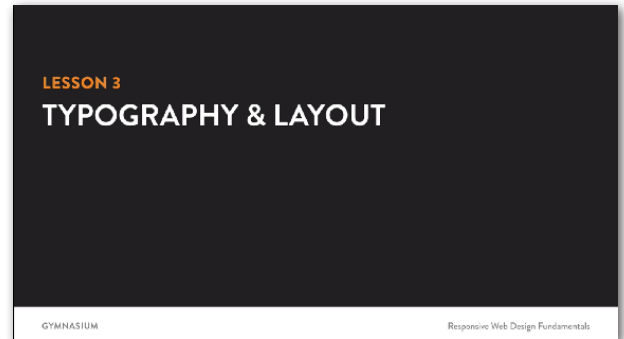
We'll start with progressive enhancement. The point's the content, and we need to make sure that it shows up. Anything else beyond that's kind of a bonus.

The history—when we look at the original HTML spec, there was actually no way to control the typeface that something displayed in it all. Eventually we had the inclusion of the font tag, and then eventually that was replaced by CSS. And there have been a bunch of advancements since, including services that allow us to embed commercial typefaces with relative ease.

But first, the content. We'll see in this screenshot it's our sample project but without any web fonts being loaded. It still works. It still functions. It may not be quite as pretty. It may not have the same tone or impart the same visual style, but it does work.

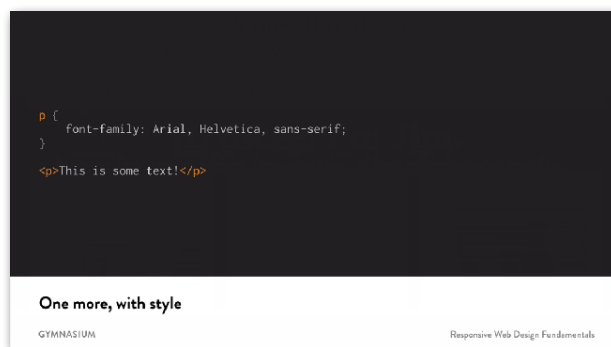
But if we go back a little ways before we could actually embed that typeface, the only option we had is make it a graphic. So the first few years that I spent working on the web, that's the only way that we could get headlines up in a typeface anything other than Times or usually Arial or Helvetica, just whatever the default was in the browser. This was the only way we could do it.

Eventually, we got this. After the first few years we had the font tag introduced, but we were still stuck with what we would refer to as web safe fonts. We could only reference things that we could be fairly certain were installed on the user's computer. So we could reference 1 of maybe 10 or 12 typefaces that we knew would be around.



Verdana was one that became fairly ubiquitous early on. Verdana and the corresponding typeface Georgia were used as a fairly ubiquitous stand-in to get past using something like Times or Helvetica or Arial.

Eventually we got CSS. So once it got around to turn of the millennium, we got the origins of CSS where we could specify a font family. And again you see this as listed in a stack. We have Arial and then Helvetica and then sans-serif because we know that on Windows machines of the time they wouldn't have Helvetica. Mac machines of the time wouldn't have Arial. And then if you are some brave soul with a Linux machine, they got the catch-all sans-serif, whatever was installed there. That was a key word for any sans-serif typeface the browser could find.



In the middle years I'd say we had a fairly dark time with sIFR and Cufon. Those were JavaScript and Flash methods for embedding typefaces, but they were really buggy, really hard to implement, and just generally sad face.

Then along came 2009. 2009 brought with us the widespread adoption of `@font-face`, which is the method by which fonts can now be safely embedded for use across the web and across every device. And in 2009 we saw the launch of Typekit. And after Typekit, we had a bunch that followed. Google Web Fonts was launched. Fonts.com for monotype launched their web font service. A couple of years later Hoefler & Co. Launched Cloud. Typography. They've got a great service now. MyFonts became really popular, a great place to go and buy fonts that you could host yourself. Many other foundries followed. But that was the point at which the tide turned, the technology started to stabilize, and we were able to bring real type to the web in a much more meaningful and accessible way.

That's enough history for now. In the next chapter, you'll begin to dive into the core concepts of responsive—

## CHAPTER 2: OVERVIEW OF WEB FONTS

So that brings us to making our typographic responsive. So in this chapter we're going to focus on performance. We're going to look at fonts and fallbacks and see how to predict what things look like during the loading process. We're going to talk about terms like F-O-I-T and F-O-U-T, and the different font-loading techniques that will help you get fonts on screen, and then come back to progressive enhancement and see how to best support it.



What you see here is the typical way to embed web fonts from Google. We have a link to a style sheet from the Google Web Font service, and then we set up our selectors to reference that font that we're loading, indicating the font family and the font weight and style that we desire. This is pretty simple, but we'll see across different network conditions this does expose us to some problems. So across a slow network connection when we're not doing anything to sort of mitigate this loading process, these

font files are assets that need to load, and the page won't render until it gets there. And this is the sort of thing that might happen. Wait for it. Almost. And there it is.

So what happens with every browser on the market right now is that there's a timeout waiting for that web font to load. Some browsers respect the timeout, some don't. Currently WebKit is starting to implement it, but in Safari's implementation, it will sit around waiting for as much as 30 seconds or longer for that web font to load and not show any content at all. That's a real problem when you're trying to make sure that you're respecting that main tenet that the content has to load.

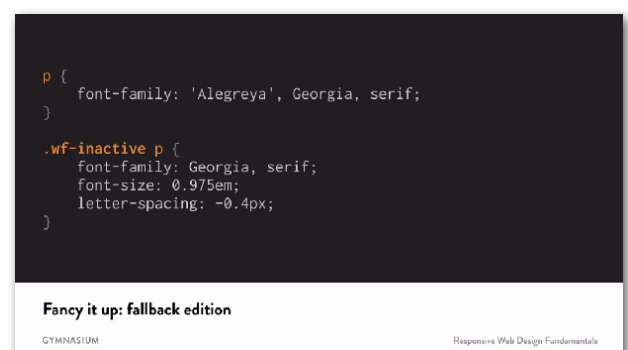
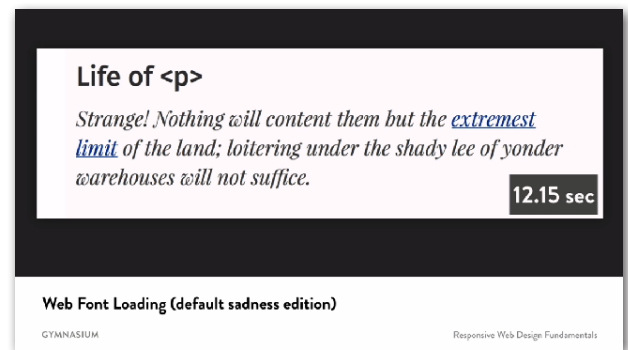
So there are a few things that we can do. Now Typekit launched in 2009. As early as 2010, they worked with Google to develop the Web Font Loader, which was a simple JavaScript method for embedding type and then ending up getting a class in your page that is injected into the HTML element, letting you know the status of the loading of that typeface. Once you have that class you can tie into it in your CSS and mitigate this flash of nothing or invisible content.

And that looks something like this. So we have this script that you can get from Google and drop this into the head of your page. And what this does is it references all of the web fonts that you want to load, but it does a couple of important things. You'll see right down below the middle, `wf.async='true'`. That's an important bit to get the content of the page rendering quickly, because that means JavaScript will fire off the request to load the web fonts and then the browser will get about rendering the page right away.

The next thing that happens is that Google will then inject a class back into the HTML element of your page and let it know that things are loading. So you can now do something like this. You have your paragraph tag that specifies the web font. And then when you have a `wf-inactive` class present, you can specify a font stack that doesn't include it. So that way your content gets on the page right away. And here's an example of taking that a bit further with your headings as well.

So with that in place, we have something that works a little bit faster, and then finally loads into the web font. So we get content on the screen a lot quicker. That was onscreen at around the 3 second mark as opposed to waiting till around 12. But we can do a little bit better.

So in order to improve on this and avoid some of the ugly reflow, which is when we start to get into the term flash of unstyled text, when we have our `wf-inactive` class present, we



can actually tune the size of the fonts and the placement and the letter spacing in order to more closely match the intended look once the web font loads.

So now that we have all of that in place, we can look at another screenshot. There's our headings again. And you'll see we got content on the screen very quickly with a minimal amount of reflow. So we had content on the screen with less than two seconds, and then we had the full rendering in around five. So that gets us a much better level of performance.

And I should note that all of these tests were done in a real browser with the speed throttled back to around 3G. Because one of the things we have to remember is most of the time when people are on a cellular connection, it's not under optimal conditions.

## CHAPTER 3: RESPONSIVE TYPOGRAPHY, PART 1

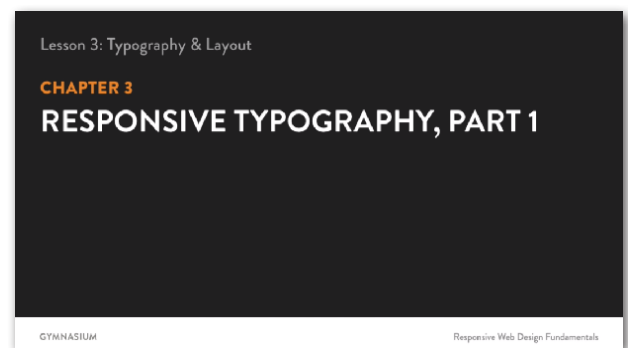
So here we are in our code for less than three. And we need to do a few things. This is in the starting file index page. You scroll down and take a look right around line 47, and you'll see here's our typical line. We're embedding Alegreya and Alegreya Sans from the Google Web Font service, and we're doing nothing else.

So we're not going to end up getting that loading class. But if we were to take a look at the final page, we'd see a couple of important things happening. Now one of the things that I'm surprising you with here is that we're actually not going to use the Google Web Font Loader. We're going to use something else. We're going to use something created by Bram Stein, who's one of the genius minds behind everything cool that goes on at Typekit. Now he created something called Font Face Observer. And that's another JavaScript library that allows you to monitor the loading of typefaces, but it goes a little bit further than the Web Font Loader does at this time and ties into some other new aspects of CSS called the Font Loading API.

So if the browser supports the Font Loading API, it does a few things a little differently, a little bit better, and it helps us monitor this loading process. And we add another little tweak to it.

So if we take a look in the final page, you'll see we have our line of CSS. And then underneath that we have a bit of JavaScript, starting here around line 48. We look at line 48 here, it's doing a bunch of things. But I'm going to skip over these first few lines, other than note that as soon as this script fires and the page is loading, we're inserting this wf-inactive class.

Now scroll down a little bit further. Skip this section for now. We look and we start to construct a reference to each one of the typefaces that we're going to add. And we construct a Promise where it looks to see if each one of these fonts that we've named has loaded. And it's using that Font Loading API and the JavaScript, and





then it has a little bit of fallback in there to see if these web fonts have loaded. If they have, we remove the wf-inactive class. We add in the wf-active class, and then the page will render the way we like.

So let's take this script and copy this. And we'll bring this back over into our start page. Excuse me, I'm going to bring it back into our working page. And we're going to bring that inside the head section. We'll make a little bit of room here. Paste in our script. So we now have the line of CSS loading from Google. And then we have our JavaScript here.

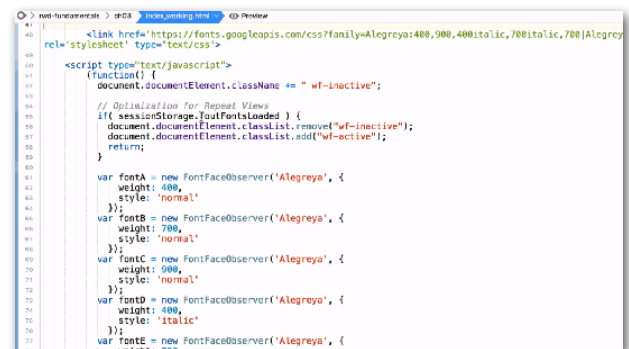
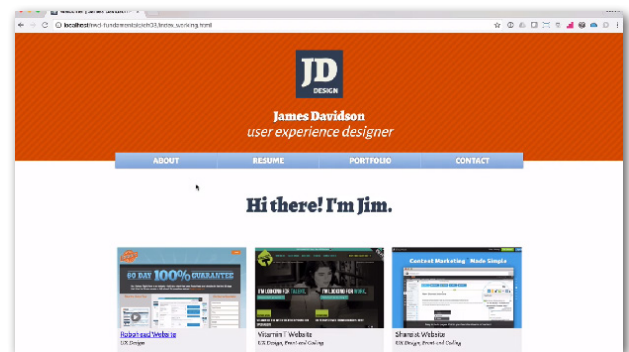
There's one thing that's missing though. We need to load the Font Face Observer JavaScript. So let's go back over here. And if we look right above, just below Modernizr, we have a line referencing the Font Face Observer JavaScript. I'll copy that. And we'll come back over to our working page and put this in right here. Save it.

Now let's go back to the browser and go to our working page. And if we throttle back our connection speed, which you can do using the Chrome Developer Tools very easily. If you look under Network, we can turn this down to regular 3G connection. Now I'm going to tuck this out of the way. Now we hit the Back button so it's the start page, and I'm going to hit Reload again. You'll notice it takes a little bit of time for everything to come in, but not too badly. And if we hit this page again with that script in place—and hit Reload. See how quickly the text comes in.

There's another little piece of magic that we got from Bram in here. The bit that I skipped over, we're actually setting a session storage variable. So that way each time we go through looking at the page, if your browser supports session storage, it's going to try saving a variable that says the fonts have loaded. One of the things that slows down browser rendering is checking to see if the things are in cache. We want to tell it that the browsers have been cached already. And if that variable is saved, then we do the class switching without having to go and do the rest of the JavaScript. So that saves a little bit of page render time.

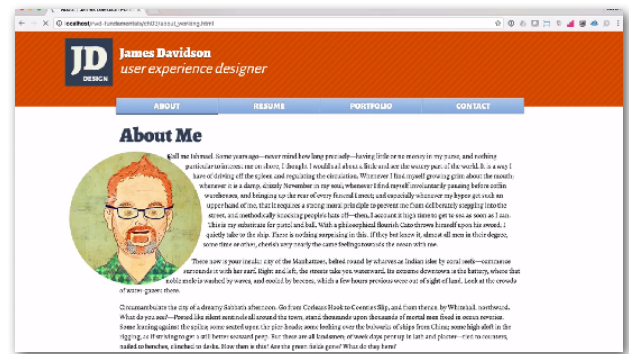
And this is the last little bit I glossed over before. When this promise fires off and it works, we get the inactive class removed, the active class saved, we then set the session storage variable. So on the first page render, it has to go through the full cycle of this script. And then every subsequent page render, it actually skips it and your page loads much, much quicker.

So we get to take this bit—and I'll end up just copying this whole section here, this whole script. And we'll bring that all the way back up to where we add Font Face Observer. Copy that. Let's bring that into a couple other pages, particularly the About page. There's a lot more text there. And we can just replace this.





So we have our JavaScript. Above that we have the call to the web font service and the Font Face Observer JavaScript. Save that. Now we click the About page. We can see even with the other assets loading, the web fonts were there right away, absolutely no delay at all. So it really does save a lot in the loading process and really gives you much better results.

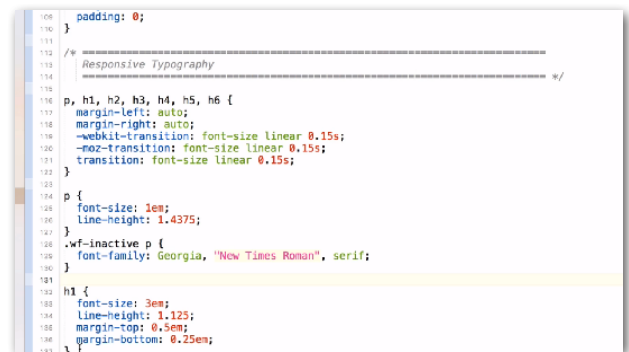


The last thing I'm going to show you in this chapter is a custom tool you can use to toggle the web fonts on and off on your pages. So this is what the final version looks like. It's a small button in the top right corner, and when it's labeled web fonts on, we're seeing the web fonts loaded. When you toggle it to web fonts off, we're seeing the backup styles being displayed.

This tool, with the associated fallback CSS, can help you do two things—one, help you understand what a user will see in that moment before the web fonts are loaded or for some reason if they never load it all; two, based on that information, allow you to tune up that fallback CSS and make sure that the font that's selected from the system, as well as all the associated stylings like line height, font size, letter spacing, word spacing, and all that best matches the finally loaded web font to avoid reflow and reworking of the design. The end result will be web typography on your site that will look its best in any given scenario. So let's add the code for the web font toggle button, and then we'll go in and tweak our default font styles as needed.

Let's take a look at the CSS. And if we start with our working CSS and we come down to where we have our type defined, you can see our paragraphs, sizes, and line height, and our heading. We're not really doing anything about the loading process, so we're not addressing any of these differences. Now we need to be able to see that difference, so we need to add in few lines of CSS here. And then we're going to add a little switch.

So if we take a look at where we're headed, I'm going to open up the final CSS and we'll grab a few bits from there. You can see in this block what we've done is add in a wf-inactive block. And that's where we can specify no web fonts.



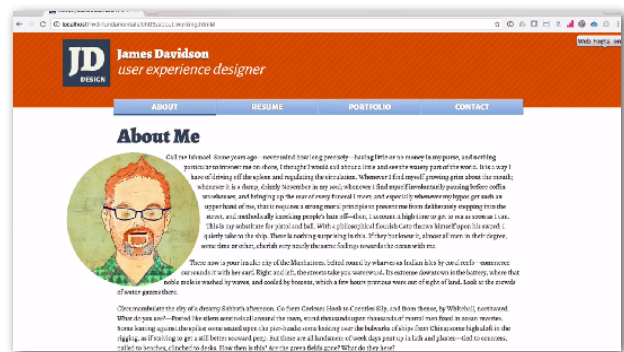
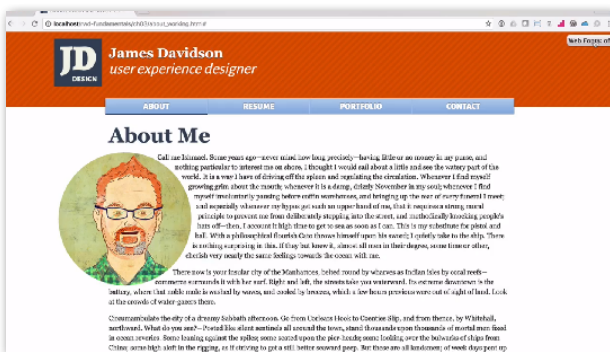
So we're going to start with that. I'm going to go back to our working CSS, and pairing these things up so we can see them together. Right underneath this block, we want wf-inactive p. And paste in our font-family line. And I'm going to do the same thing with the heading.

And now we need a way to see what it looks like during that loading process. So I created something to help us out. And that is a little button that we can place. And we're just going to grab this whole chunk of code from the beginning to the end of the web font toggle, and we're going to place that in just inside the body tag. So we pull up our working file. Just inside the body tag, paste in our button. Save that and go back to the

browser. And that was in index working. So I'm going to back to the main page. And that loads. We see we have a little toggle here.

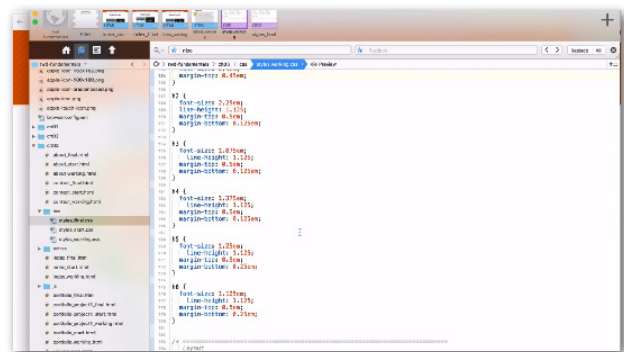
Now what this is going to do is actually just toggle that class active and inactive. So I turn it off. You can see it drops back to—in the h1 and the body text, you can see it goes back and forth between showing the body copy as we'd expect it.

Now let's add that in the About page where it's a little bit more text-heavy. We go to our about working page, and look right inside the body tag. Add in our switch. Now let's go to the About page and toggle these on and off. You can see that this text reflows quite a bit. So we clearly want to do a little bit to make sure that this reflows nicely.



And if you look in our final CSS, we actually do address that by adding in font size, letter spacing, and line height. So let's grab a little bit more. So we'll be correcting our paragraph text. And then we're also going to grab a little bit here.

Now you'll notice that I am getting this from a little bit different spot. I'm going to come back to that in a minute. But we really just want to make sure we get the font size and margin settings the way we want them to be. And so if we reload this page and turn this off and on, see how much less reflow there is? We have the text that sits in roughly the same place. We've only got one or two words that bump from one line to the other. And the h1 occupies a much more similar amount of physical space, the margins above and below. Though we can't get it perfect because there are very different metrics in the typeface, but you can minimize the amount of difference that you see. And if you do that across all of the main type sizes—so the headings, your navigation, the text in your header up here. Make sure you've got all those covered so that things occupy roughly the same amount of space. It really does help minimize what people will notice during the loading process. You can see again as we flip back and forth, even with the speed throttled down, because we have that variable set, we end up getting the web fonts loading really, really quickly.



Now I'll leave this to you to go and add this back into some of the other sections. So we'll want to do this

for each of the headings. And the next section we're going to get into adding to this a little bit and not only respond to the presence of the web fonts but also respond to the device itself.

So here's where we should be by now. We've added JavaScript font loading. We'll do that to all the pages in the site in that section. We've added the loading class to minimize flash of invisible text and make sure that you're that loading class, we get text on screen quickly. We've added some styles to minimize the differences between the fallback and the final fonts. And this is our mission of font mitigation. And then we ice the cake a little bit with that JavaScript trick to minimize that reload time by using a session storage variable.

And with all those things in place, especially with that toggle switch to help speed up the process, you can really improve the user experience across their time on your website by taking care of these things. And it will pay dividends for anybody that's coming to the site on a fast connection or a slow one.

## CHAPTER 4: RESPONSIVE TYPOGRAPHY, PART 2

Next chapter—responsive typography, part two. In this chapter, we're going to build on the responsive typography concepts and start thinking about proportion—and really examine that phrase, content first, and what that means for your typography. The basic lesson that we want to take away is what works in print works in print. It doesn't necessarily work everywhere else. So let's take a look at this layout—looks perfectly fine on the desktop.

But when you make it shrink down to a small screen, you can see that exact same proportion really looks, well, ugly. The important thing on the screen here is the title and the main heading. We know that those need to be the most important and prominent things on the page. But clearly, they don't need to be quite this big. You still get the point here by scaling them down, allowing for a better line length and less wrap, while still maintaining hierarchy.

Now, the reason this works is that there are fewer things on the screen. The smaller the screen, the fewer the elements, the more subtle you can be in the transitions. So that's what we want to tie into—in the same way that we tie-in to media queries and viewport sizes to adjust lay out, we can also use them adjust type. Now, you don't want to forget about your emails. Or you end up with things like this.

It's great to have their layout respond to different viewport sizes, because after all, something around 75% of all emails are opened on a mobile device first. But when you don't take into account the differences in screen size and word length, then you really end up with some big misses in the way things lay out. So this table is something that I started to work out a few years ago. And it really became a central part of my book.

And it really led me to thinking about this as responsive typography in the first place. So we have in the first column, our values that came right out of Robert Bringhurst's book, Elements of Typographic Style. And these



are really time honored traditions mapped to web constructs—so main headings usually being set around three times the size of the body copy, subheadings set in smaller proportions. But when you look at that map to the web, we make some small adjustments from 12 point type to 16 pixels or 1 Em.

And we start to see that we can use those same proportions on large screens. But as it gets smaller, we really want to adjust. So I started working out of scale. And this came through an article that I wrote for the Typecast blog a while back. And that scale really helped influence my thinking about how all of this should work. And so while these aren't absolutes, these are good numbers to start with, as you think about how we bring proportion into our work as we're adjusting layouts for screen size and device capabilities.

So let's go back to our code. Now, in this code section, we see under the responsive typography section—while we're waiting for this to come along—we have our basic definitions with some of the fallbacks for paragraphs and headings. But we don't really have anything to respond to screen size. So we want to really think about how that might work. And knowing that we're starting from mobile first and working our way up, if we cheat a little bit and take a look at our final styles—if we look at the H1 declaration, we have H1, the inactive H1.

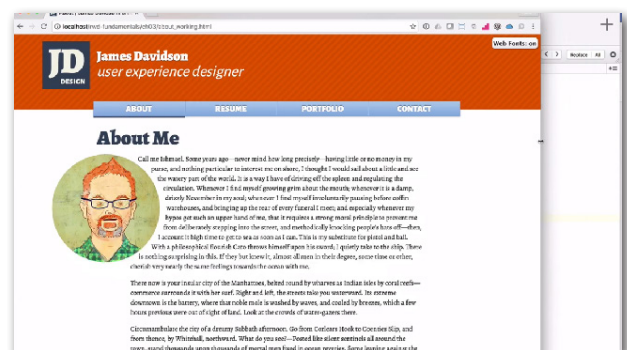
And then we have breakpoint definitions for minimum width 25 Em, 43 and 3/4, 56 and a 1/4, and 81.25, all the way up to the largest desktop sizes. And you can see in each instance, we have definitions for font size and line height. That's going to be slightly different across each breakpoint. And then we have the corresponding corrections for all of our headings that would go along with it during the loading process. So if we were to bring this whole section in, we'd have this addressed across a whole range of screen sizes.

```
158 @media only screen and (min-width: 25em) {
159   h1 {
160     font-size: 2.25em;
161     line-height: 1.25;
162   }
163   .wf-inactive h1 {
164     font-size: 2em;
165     line-height: 1.25;
166     letter-spacing: 1px;
167     margin-top: 0.65em;
168   }
169 }
170
171 @media only screen and (min-width: 43.75em) {
172   h1 {
173     font-size: 2.5em;
174     line-height: 1.125;
175   }
176   .wf-inactive h1 {
177     font-size: 2.4em;
178     margin-top: 0.5em;
179   }
180 }
181
182 @media only screen and (min-width: 56.25em) {
183   h1 {
184     font-size: 3em;
185     line-height: 1.125;
186   }
187 }
```

So we're going to do that. But first, I want to bring you back to the paragraph. This is the one element that I really recommend you rarely change. Body copy set at 100% or 1 Em, in this case, is going to be universally readable across all devices. And that's really important to keep in mind. When you set something in pixel sizes, not only are you unable to resize it in some older browsers, and actually in the current shipping version of Chrome due to a—not sure if it's a bug or a feature.

But it's really important to know if you set this in 16 pixel size, you would not be able to resize that in one of the most popular browsers as of today. So we set this in a relative unit, 1 Em, and we leave it alone. The only thing that we might do is, as the screen gets really big, we might bump it up a little bit. But in general, this is going to remain a constant. We know it will be a good, readable size across any device. It's only the headings that are going to adjust in relationship to what's going on in the paragraph.

So I'm just going to grab the paragraph declaration, because we have the larger screen size, and then I'm going to come down and grab the heading—everything for H1. And we'll bring all of this into our CSS. Save that and go take a look



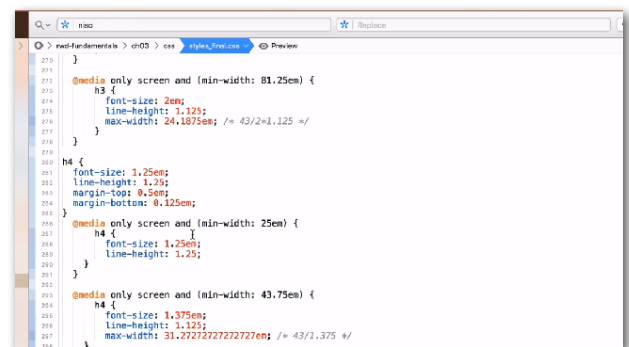


again. Now, when we do that, we can see that we're wide, so the headings looking a little bit bigger. As we start to bring that screen in, you can see that text size change for the headings. Body copy always stays nice and readable.

So we have these definitions set up for each of the different levels of heading. You can see these defined in your final file. I'd recommend just grabbing some of these, so you don't have to write it all out. But it will—it pays to kind of take a look and see which breakpoints we're trying to address this with. And really when we go back to this table, here's where this starts to make a little bit more sense.

You can see starting from the right, our mobile sizes. Then you get to minimum width of 25 Em—we're going to deal with small tablets, and then slightly larger for the larger tablet. And you can see body copy stays the same all the way across. And the headings start from about 2, and work their way up to 3. You might go further, but it really depends upon the design and the typeface itself.

The important thing to note is that each one of those moves up in a relatively predictable rhythm from that slightly larger than body copy, up to that optimal size in your typographic scale. So go ahead and bring all of these styles over, so that you have all of the corrections in for the loading process and the media queries. And bring those in. And then work your way through the rest of the HTML working files to add in the loading JavaScript and the little toggle button. So you can click around through your site, and make sure that everything's loading and scaling properly.



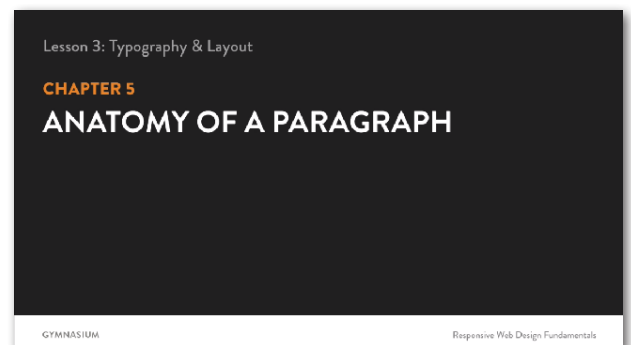
```
275 }
276
277 @media only screen and (min-width: 81.25em) {
278   h3 {
279     font-size: 2em;
280     line-height: 1.125;
281     max-width: 24.1875em; /* 43/2=1.125 */
282   }
283 }
284
285 h4 {
286   font-size: 1.25em;
287   line-height: 1.25;
288   margin-top: 0.5em;
289   margin-bottom: 0.125em;
290 }
291
292 @media only screen and (min-width: 25em) {
293   h4 {
294     font-size: 1.25em;
295     line-height: 1.25;
296   }
297 }
298
299 @media only screen and (min-width: 43.75em) {
300   h4 {
301     font-size: 1.375em;
302     line-height: 1.125;
303     max-width: 31.272727272727em; /* 43/1.375 */
304   }
305 }
```

## CHAPTER 5: ANATOMY OF A PARAGRAPH

Now, when you're thinking about your typographic style, I recommend that you really start with a paragraph. Start with that smallest unit of content. Look at your line length. Look at your line height. Look at the text size, the length styles—everything about this one unit of text—and build everything from there.

When you start here and build everything out, you've got a nice solid foundation. And the rest of the typographic system will scale from there. So I'm going to grab all of this CSS and bring it into our working file. And then we're going to start to add and a few other touches. So we got all of this.

Replace these heading declarations. Now, what we want to think about is, what is a sensible line length? So one of the things that you'll notice here with all this proportion is that the line length can get pretty long. And that's a problem. So when we're thinking about that perfectly laid out paragraph, we want to think again about an optimal line length for reading.



And one of the things Robert Bringhurst wrote about, in *Elements of Typographic Style*, was an optimal line length being—well, he wrote somewhere between 66 and 75 characters. And that can be quite a bit shorter than what we see here. As layout scale and responds to design, it's quite often you end up with line lengths that are really quite long.

Now, this may not bother some people aesthetically. Although, I find it looks a little weird. The real issue is that the longer the line length, the more tired your eyes will be and the harder it will be to get from the end of one line back to the beginning of the next, and not lose your place. So what I typically do is actually add a maximum width in for that paragraph.

So if we come back to our CSS and write in here in the paragraph definition—and this will apply across the board. I'm just going to use a simple max width. Now, you can experiment with the value that you use here. But somewhere around 40 Em will usually get you a line length that's pretty good. I'm going to try 43 Em.

Save that, go back here, and reload the page. And you'll see you have a much easier line length to read. Just brought it in. It's probably 30—or maybe 30 characters or so less. But it's a little bit longer than you might see in print. But it really does work nicely in proportion. And that's something that, as you scale down, it may fill up the width of the column a little bit more, when you look at it in relationship to the navigation bar above.

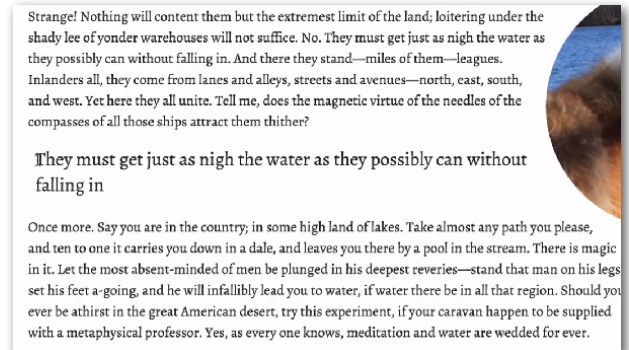
But then as the screen gets larger, it doesn't end up getting any longer. So it preserves that readable line length even as your layout expands. Now, there's another thing that we can do to play around with block quotes, if we want to set those in a slightly different style. And we can highlight those a little bit better. So I'm going to add that in at the bottom of our typography section.

And we'll just use the block quote tag. And we'll try something like setting that in a slightly different typeface. So I'll use our similar fonts stack, and set it—Alegreya, Georgia, serif. And we'll set the size a little bit larger. And I'll set a max width—I'll just try 80%.

We'll set it in a little bit, and set the margin. Nothing above, auto for the right, one and below, auto for the left—so that will keep it centered inside the text area. We'll save that. And we'll go back in our page, and we'll just take one of these phrases. And we'll copy that over.

So we'll take this, copy it, make a blockquote. So here's our blockquote. That's set here. And doesn't really stand out that much yet. So we'll go and add a little bit more. Maybe we'll set text style. Oops, excuse me—font style. Italic.

And instead of 80%, let's actually set the max width—we use the same units that we're using above. We'll set this to be more like 20 Em. Let's see what that looks like. And remember that it's 20 Em based on the font size, which is slightly larger. So we do have that little bit of inheritance math



going on there.

But here's our block quote set apart—nice italics. And you see that will scale along with everything else. So these are things that you can do to polish up your typography a little bit—tie into those same kinds of rules that we have for adjusting based on screen size, and give it a different level of prominence.

We can play around with adding media queries in there if we wanted to, and have a little bit of fun with it. This also gives you the chance, with block quotes, to do things like adding large quotes in the before element, or just change that style, so that it stands out a little bit more on large screen, but maybe is a little bit more subdued on small. So in this chapter, we've added styles for a range of screen size for all your main elements.

We've set styles for line height and maximum line length. And we've added some styles for quotes with an alternate typeface, with a little bit more fallbacks as well. So all of these things taken together, along with the loading practices that we worked in the chapter before, really add up to a very robust typographic system that's going to work across really anything that you throw at it.

## CHAPTER 6: LAYOUT EVOLVED: CSS COLUMNS & FLEXBOX

In this chapter, we're going to evolve our layout. We're going to get a few refinements in here. We're going to look at some of the history of how layout has evolved in print and how we can start to bring some of that level of polish to the web. And we'll add some of these features to our project.

So what I have on the screen here is a demo from Jen Simmons and her lab's website that has an example of what we typically see—no columns, just one big wide paragraph of text. And you can imagine that, in our project, this would certainly benefit from a maximum line length. And when you look at that in a responsive site, it's typically just made narrow.

But you can use CSS columns. And those are pretty widely supported. So as you scroll down, you can see that it's broken seamlessly into two columns. If we reflow, you can see the content goes smoothly from one column to the next. You can specify more and more columns with a maximum width. And you can even do this with images and rules to really start to mimic a newspaper layout, bearing in mind that you need to be cognizant of the height of the window. So this is where you might be able to do something interesting looking at a vertical media query or even using vh units it's like we mentioned in the last chapter.

But how might we use this in our project? Let's go take a look at a couple places where it might benefit from

But look! here come more crowds, pacing straight for the water, and seemingly bound for a dive. Strange! Nothing will content them but the extremest limit of the land; loitering under the shady lee of yonder warehouses will not suffice. No. They must get just as nigh the water as they possibly can without falling in. And there they stand—miles of them—leagues. Inlanders all, they come from lanes and alleys, streets and avenues—north, east, south, and west. Yet here they all unite. Tell me, does the magnetic virtue of the needles of the compasses of all those ships attract them thither?

*They must get just as nigh the water as they possibly can without falling in*

Once more. Say you are in the country; in some high land of lakes. Take almost any path you please, and ten to one it carries you down in a dale, and leaves you there by a pool in the stream. There is magic in it. Let the most absent-minded of men be plunged in his deepest reveries—stand that man on his legs, set his feet a-going, and he will infallibly lead you to water, if water there be in all that region. Should you ever be athirst in the great American desert, try this experiment, if your caravan happen to be supplied with a metaphysical professor. Yes, as every one knows, meditation and water are wedded for ever.



Lesson 3: Typography & Layout

### CHAPTER 6

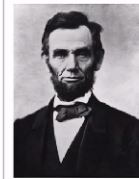
## LAYOUT EVOLVED: CSS COLUMNS & FLEXBOX

GYMNASIUM

Responsive Web Design Fundamentals

### Columns With a Rule and an Image

If we could first know where we are, and whither we are tending, we could then better judge what to do, and how to do it. We are now far into the fifth year, since a policy was initiated, with the avowed object, and confident promise, of putting an end to slavery agitation. Under the operation of that policy, that agitation has not only not ceased, but has constantly augmented. In my opinion, it will not cease, until a crisis shall have been reached, and passed.



I believe this government cannot endure, permanently half slave and half free. I do not expect the Union

'A house divided against itself cannot stand.'

to be dissolved—I do not expect the house to fall—but I do expect it will cease to be divided. It will become all one thing or all the other.

Either the opponents of slavery, will arrest the further spread of it, and place it where the public mind shall rest in the belief that it is in the course of ultimate extinction; or its advocates will push it forward, till it shall become alike lawful in all the States, old as well as new—North as well as South.

Have we no tendency to the latter condition?

Let any one who doubts, carefully contemplate that now almost complete legal combination—piece of machinery so to speak—compounded of the Nebraska doctrine, and the Dred Scott decision. Let him consider not only what work the machinery is adapted to do, and how well adapted; but also, let him study the history of its construction, and use, if he can, or rather fail, if he can, to trace the evidence of design and concert of action, among its chief architects, from the beginning.



evening out the layout and creating a little bit more visual interest.

We come back over here. I've loaded up our sample site. This is the portfolio project one working file. And if we look here and scroll down, it's OK. It's fine. But maybe we could do a little bit more. And when we go out to the portfolio main page, one of the things that makes this layout look so nice is that we have this uniform grid. But when one title is longer than another, you can see that starts to break down. See this center box extends a little bit further down than the ones on either side.

Well there are a couple of techniques that we can work with that will help make that look better. Flexbox is something new in CSS that's now being pretty well supported. And it falls back pretty nicely if it's not present. So we can really think of this as an enhancement.

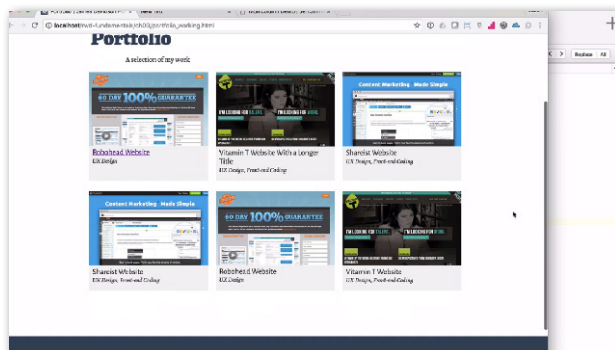
If we look at the way this layout is done—let's go ahead and take a look at our code. And if we look for the card layout, card groupings, it's a little bit further down in the file. If you take a look down, it's right around line 1,133 or so, we look at the card-grouping. And you can see that we have some CSS in here that governs the width of those card boxes. And it sets a clearing based on n-th child selector to make sure that things wrap nicely as the screen size changes.

```
1130 margin-left: -1.5%;
1131 margin-right: -1.5%;
1132 }
1133
1134 .card-grouping .card {
1135   display: block;
1136   margin: 0 1% 1.5em;
1137   width: 100%;
1138 }
1139
1140 @media only screen and (min-width: 25em) {
1141   .card-grouping .card {
1142     float: left;
1143     width: 48%;
1144   }
1145   .card-grouping .card:nth-child(2n+1) {
1146     clear: left;
1147   }
1148 }
1149
1150 @media only screen and (min-width: 56.25em) {
1151   .card-grouping .card {
1152     width: 31.333333%;
1153   }
1154   .card-grouping .card:nth-child(2n+1) {
1155     clear: none;
1156   }
1157   .card-grouping .card:nth-child(3n+1) {
1158     clear: left;
1159   }
1160 }
```

But what we'd really like to do is actually change this display from block to flex. Now we only want to do this if it's actually going to be supported. So this is where Modernizr comes in. Now one of the tests that we have in our Modernizr file that we'll look at what's happening on the page is whether or not flexbox is supported. So we notice here in the HTML page we have a flexbox class that's been inserted. So that's the way we can scope this new CSS and make sure that it only applies to browsers that will actually understand it.

So we look at our code. Again, we're going to cheat a little bit. We're going to go over to the final file. And I'm going to look for card-group. That's around 1,175 in this file. And if we look down below that main chunk of CSS, now we have some that's scoped behind the flexbox class. So if we grab this, we can see that this card-grouping, the grouping itself is set to display flex. And we have a webkit prefix in here to make sure it works in as many places as possible. And we set the flex-flow to use rows and to wrap. And the wrapping is important because we're going to tell it how wide to make these cards. And that way we know it'll wrap one row to the next. And justifying the content space-around just means that we'll end up having nice even padding around. One thing you'll notice is I've set the margin on the left and the right to minus 5% because I've added a little bit to offset that padding so that things still line up evenly with the content above and below.

So we're going to grab this whole section, copy that, and bring it back over to our other file. Insert it right here above card. And if we save that and go back to our portfolio page, now I'll leave it at this width. So if we've done it right, then these boxes are going to even out. And there we go. We have even-height boxes all the time that will still reflow as we'd like them to and preserve that equal height with anything that's



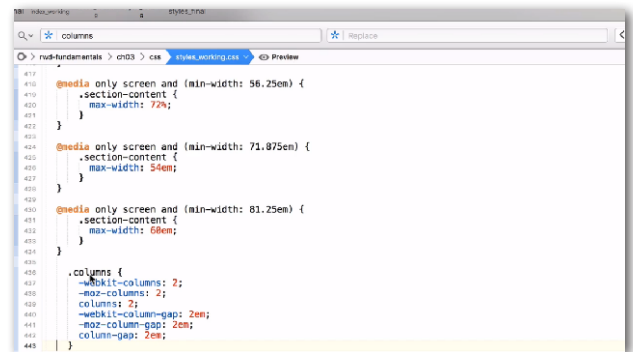
next to each other. So whenever they're next to each other in the same row, they'll always be an equal height, and each row will adjust to its tallest piece of content.

So that's one place where we'd use it. And we'd probably also end up using that on the home page. But because we've added it and scoped it to that card-grouping, and we're really smart about how we've added these things, even if we were to go and add a really long title to the middle here, this should still pick up those styles.

So why don't we test that out? I'm going to go in here, go to our index working page, and I'm going to find that card. So here's our cards here in the middle, "Vitamin T Website with an extra, extra, extra long title". Save it. Go back and reload. And you can see it picked up those styles right away because we're reusing those content modules across the site. And once we set the styles once, they'll be picked up everywhere else.

Now the other thing we wanted to try was spicing up the layout here a little bit and trying to add a little bit of visual interest. So we're going to try adding some columns and maybe break it out on the left and right a little bit to create a little bit more visual interest and start to mimic a little bit more of those print techniques that break up the layout of the page.

So if we come back over here, we're going to look at our CSS. The column syntax is really simple. So if we go over here, search for columns—OK, so I've grabbed just that core CSS. I'll make a little space in here and add that in. So we have our basic CSS outlining what do we want to do with columns. We've specified a number of columns and we've specified a gap. So any place where we add this class of columns, we know that it will work out this way.

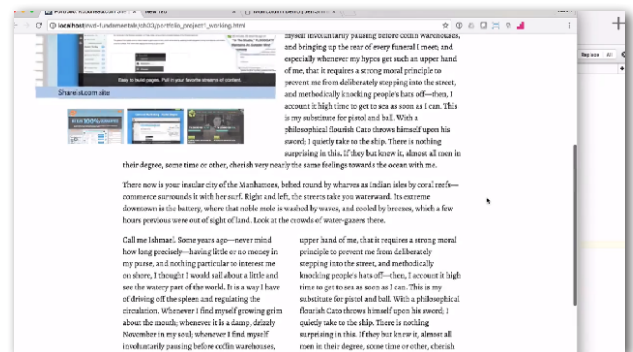


Now we've added the styles. We still want to go back to that portfolio working page. Now all we have to do is pick one of these paragraphs and add the class, which we've already done. I've added this in here. So now that we have the CSS, we should be able to just reload that page. And there we have our two columns.

Now as the screen gets bigger and smaller, we still have our columns. But that presents a problem right there. We actually might not want to have those columns on a really small screen. So we probably want to go back and look at those media queries and decide at what point we want to break this text into separate columns.

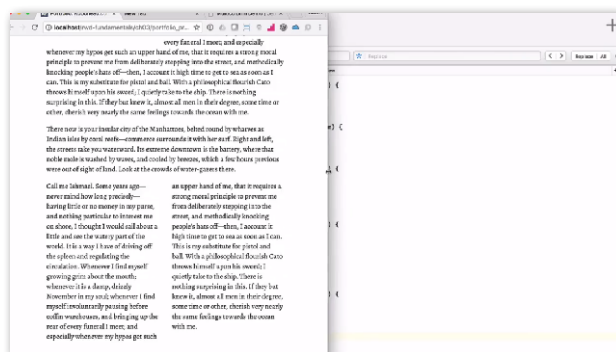
Let's go back over here, and we'll grab this media query. So we're dealing with a larger tablet-sized screen, roughly. We'll paste that in. We're going to add our own content there. And we're going to take this.

Now since we don't actually want it to do anything on the small screen, if we cut it out and leave that selector empty—just to remind us that it's there, so we keep everything grouped together. So now that we have this set up in the



media query, we have nothing out here. So in the smallest screen size it shouldn't do anything at all and should only break it into columns once it reaches that larger screen size. If we've done this right, when we refresh this page, the columns should go away. It's back to a single column until we get to that wider screen, and now it breaks nicely into those two columns.

But if we wanted to do this to actually create a little bit more visual interest, we might want to think about breaking it outside the bounds of that column. So what I ended up doing here was actually just take the max width away. So that will allow the paragraph to be as wide as the content area even when the paragraph maximum width is slightly narrower. So we're going to bring this over and add that in. So once we get onto those larger screens, it takes away the maximum overall width. And we reload. And we see it comes out to the natural widest point in that content area. And then as the screen gets narrower, it just respects the same border as the rest of the content area and collapses back to a single column on smaller screens.



So it helps you create a little bit more visual interest, some shorter line length for some quicker reading. But just recognize that you want to be mindful of how tall this content is based on screen size so people don't have to end up scrolling up and down to go back and forth from one column to the next.

So in this chapter we've converted our gallery of flexbox, and we still have the fallback if flexbox isn't supported. And we've added columns to display on the project description on larger screens so we can create a little bit more visual interest on longer passages of text.

