GYMNASIUM

RESPONSIVE WEB DESIGN FUNDAMENTALS

Lesson 5 Handout

Advanced Enhancement

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. Although you can serve a single image on your website and scale it responsively, in many cases this is not ideal. For example, users with a small screen device such as a smartphone do not need the same high resolution image that a user with a large desktop monitor needs. Forcing the small screen user to download a high resolution image (especially on a slow connection) could result in abandonment of your site before it loads.
2. On many cellular networks users pay for bandwidth costs and therefore serving them unnecessarily large images could end up penalizing your visitors directly.
3. Based on feedback from the web community, there have been two recent additions to the HTML specification: the **<picture>** element and the **sizes** and **srcset** attributes (both of which can be added to the <img> tag).
4. There are two common scenarios where you might want to use these new responsive image solutions. The first scenario is *resolution switching*, defined as selecting a different source of the same image because you need a different resolution of that image.The second scenario is *art direction*, defined as modifying the content of an image under certain conditions (such as cropping for a small screen).
5. Element queries (also called container queries) are an experimental technology that allows content to react based on its own size (rather than the size of the browser window). For example, if you wanted to adjust the font size of some text based on the number of characters, element queries could help with that. To use a version of element queries today requires a JavaScript library, and is not recommended to use in production, however it is concept worth paying attention to down the road.
6. Designers looking for ways to create unique and innovative page layouts must balance new technologies with web browser support. One new technology that has arguably reached the requisite level of stabillity is CSS Grid Layout, a comprehensive collection of CSS properties that allow you to quickly build complex page layouts with fewer lines of code than traditional layout techniques.

ASSIGNMENTS

- Quiz
- In the Lesson 5 folder there is a document labeled “responsive_images_assignment.html”. Additionally there are two sets of images, one set has a number of images labeled “misty_sunrise” and the other set has a number of images labeled.”tristan_and_tillie_landscape”. Choose one of these two sets of images and convert them to responsive images using the techniques covered in this lesson.

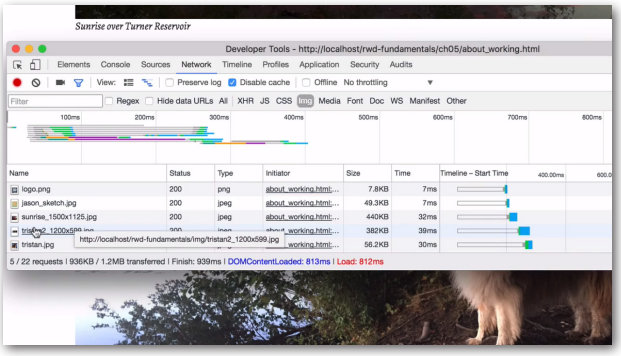
INTRODUCTION

(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.)

In lesson 5, we're going to talk about taking responsive design to the next level. We'll look at some of the challenges involving responsive images and how to make images just as flexible as our layout and text, while also keeping an eye on how images will affect the size of the page, and how to avoid penalizing your users with inappropriately sized or optimized images. We're also going to discuss the role of element queries, which are an up and coming addition to CSS. This topic of element queries will lead us naturally into our final discussion, the future of responsive web design, and specifically some areas of technology that I think will be worthwhile for you to look into.

CHAPTER 1: RESPONSIVE IMAGE GENESIS

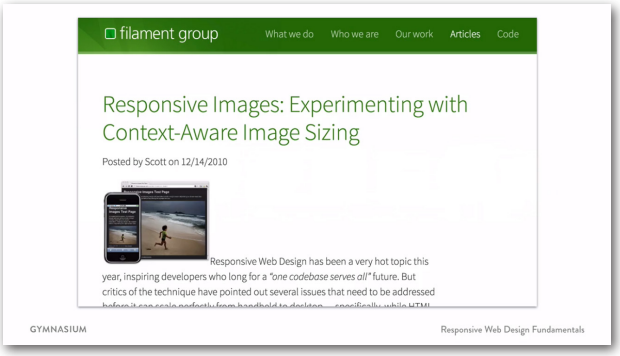
Looking at where we came from. We're going to cover a bit about the limitations of just scaling the images on the screen and look at the evolution of the different approaches and what we have as a new standard. They've added in a couple of images that will highlight what we're going to work with. And you can see on the screen—large screen and small—we've got this image of my pal Tristan, and that same image is just being scaled as we've been scaling everything else. So that same 393k, 1200 pixel wide image is being delivered to both the phone, as well as the desktop.



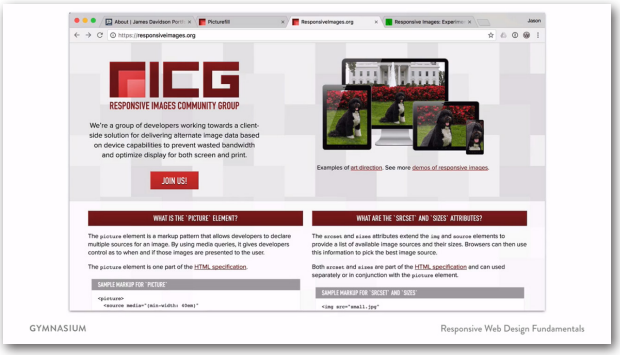
Now it looks beautiful, but we probably don't need a 400k, 1,200 pixel wide image on a phone. So we try to find some way to deliver a more appropriate asset. In the About page, I've added a couple of photos. I've got a photo of the sunrise, and scroll down a little bit further, we have that same picture of Tristan. And what happens as we scale the page, you can see that the image does work nicely in the layout, but we haven't done anything about the size of the resource.

And if we inspect this and take a look, we'll see that as we take a look here and reload and look at the images here, we are loading in these two images, one's 440k and one 382. Now that's an awful lot of extra stuff to be forcing down onto a really small screen device when they're not even capable of displaying all of that detail. It looks great, but it's not really going to be that helpful to the user.

So let's take a look at some of the solutions that have come up over time. Now the Filament Group was a big part of the origins of responsive design and the big reveal in the Boston Globe website that Ethan Marcotte worked on with them and with Upstatement. And they knew this was going to be an issue. So they came up with a JavaScript polyfill essentially that enabled us to do something simple like add into our image tag a data resource that would indicate the larger size image. And then on page load, that JavaScript would detect the size of the window and replace the source of the image from the small to the large.



Now the fallback here is that you still have a standard image that's being loaded. So if the JavaScript doesn't load, you still get something on the screen, but it also means that on larger screen devices, there's always going to be two images loaded. It's going to load the small one and then it's going to swap out to get the large one once the JavaScript loads. So it was not a perfect solution. So over time, some of the people involved in that initial site started the Responsive Image Community Group. And this group ended up doing a lot of research to figure out what was needed in order to create new HTML elements, as well as a way to style them with CSS. And eventually, the results of this effort were adopted by browser manufacturers and we can use them today.



To make a long story short, here are the two main features that came out of the Responsive Images Group and made their way into the HTML specification. The first is a new picture element, and the second are two new attributes that can be added to the familiar image tag. And those two attributes are sizes and srcset. We're going to look at how to use these tags as we continue to work on our project in the next chapter. So let's just focus on the abstract level because there are essentially two scenarios where you want to use them.

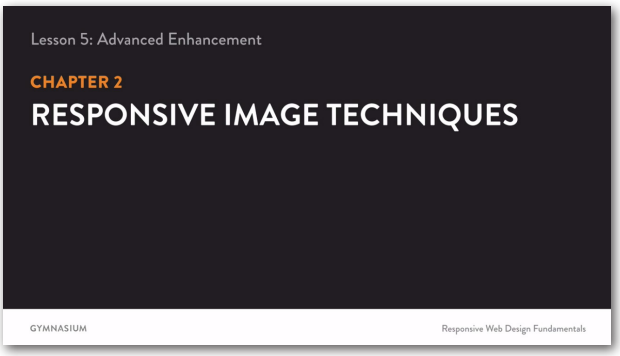
The first scenario is resolution switching. Resolutions switching is defined as selecting a different source of the same image because you need a different resolution of that image. There are a few reasons you might want to switch out the same image, but the size of the user's screen is the primary reason. The classic example being the one we discussed earlier. A desktop user with a huge monitor can take advantage of a high resolution, very large image and see lots of detail, but with a smartphone user using a small screen, the detail in that image is wasted. So we switched to a lower resolution image, which appears the same, but has a smaller file size. Another example is using the srcset attribute for higher DPI devices, like iPhones.

The second scenario is art direction. Art direction is defined as when you need to modify the content of an image under certain conditions. What do I mean by the content? Well, imagine you have an image that works great on a large screen, but the detail gets lost when it gets shrunk down to put on a phone. So the solution would be to actually crop that image down to just the main focal point when it's displayed on that small screen, and that can solve a lot of problems.

So these are the two main scenarios, and I'll give you my best practice advice now, and we'll explore both of these further in code in the next chapter. Generally speaking, if all you're going to do is resolution switching, use the sizes and srcset attributes. If you're solving for art direction, then use the picture element technique. So we've taken a look at some of the challenges that we've had over time. We've looked at a couple of new approaches, and now we're going to actually explore these emerging solutions and actually incorporate them in your own project.

CHAPTER 2: RESPONSIVE IMAGE TECHNIQUES

In this chapter, we're going to look at the two scenarios we discussed in the last one, but this time use our actual project files. The first thing we'll explore is the resolution switching scenario. And for that we'll be using the SIZES and SRCSET attributes to allow the browser to choose the best image based on the width of the viewport window.



The second scenario will be Art Direction. And for that, we'll be using the PICTURE element to force the browser to use specific images to load in order to better match our layout. After that, we'll need to spend some time discussing and adding browser support and Polyfils to these two techniques as needed. So here's a look at the image tag with sizes and srcset in a little bit greater detail. So let's take a look at what's going on here.

Now sizes is meant to give the browser some cues about how you will be using this image once it's been determined which is the appropriate one to show. So we're basically saying that when the screen is at a minimum width of 36em, we'll probably be displaying this image no more than 80 viewport width units wide, so about 80% of the window width. Otherwise, we'd be referencing a 100 viewport width units, or 100%, in terms of determining how wide an image we want to cover that screen size.

Now if you look in srcset, we list them from small to large, we have the smallest image, and we just give a hint, 500w is 500 pixels wide. So the browser can look at this list of images—it's just a comma-delimited list—and know what the pixel with is so that when it's determining the usage of that image based on the overall width of the window, it will pick the right one in factoring in pixel density as well. And you still have that source fallback, so if srcset and sizes is not understood, it's still going to pick an image here.

It's worth noting that this source is your fallback. For any browser that doesn't understand the previous bits of code, it's going to pick this one. So if you want to optimize for the smallest screen, you'd simply substitute this thousand-pixel-wide version for one that's a little bit smaller. And your default will be that small image.

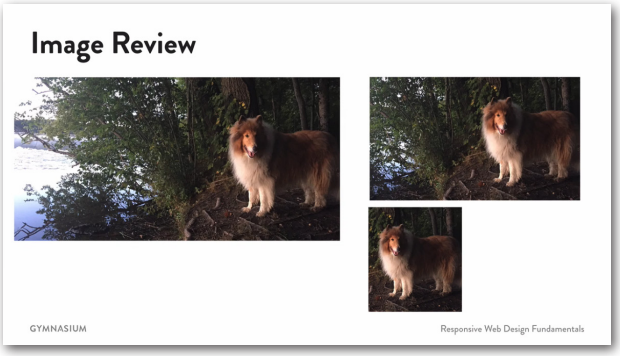
And it's important to note here that because this is baked into the browser, we don't have to worry about duplicate downloads. The browser is smart enough to only load the one asset that's correct and supported for this particular browser. Here's a look at the picture element in greater detail.

So we have three different sources defined. Inside, we have the 1,200-pixel-wide version, which we will display with a media query for a minimum width of 801 pixels. So as soon as we get bigger than the last one, we'll swap this image in. The next one is an 800-pixel-wide version, and we set a media query and min width of 501. And then we have a 500-pixel-wide version, which is going to be used in any other case.

Now what you'll also notice is that these are very different aspect ratios, and we're still picking that same image fallback. So again, you'd want to choose the image that most closely targets what you want as your baseline in case nothing else loads. I have the largest image in here, but you could just as easily put the smallest. So let's take a look and see what this looks like in action.

Before we dive into the code, let's take a quick look at what all the images we'll be working with look like just to help you visualize them. The first image you'll be working with is the Sunrise 1,500 by 1,125 JPG file, which looks like this. In the Images folder, there are three additional images, named Sunrise 500 by 375, Sunrise 780 by 585, and Sunrise 1,000 by 750.

Note that all of these images look exactly the same. So that's your clue that we'll be using these to do some resolution switching. So this is the Tristan 1,200 by 599 JPG, this is the Tristan 800 by 467 image, and this is the Tristan 500 by 557 one. Now as you might guess, these are all cropped a little bit differently, so these are the ones we'll be using in the picture element.



Now that you've got those in mind, go ahead and fire up your code editor and find your lesson file files folder. And then go ahead and open up the About Working file and scroll down to approximately line 200. And we're going to take a look down here. We get into the text of the page, and you can see we have our two figures in here, but they have only that basic image in there.

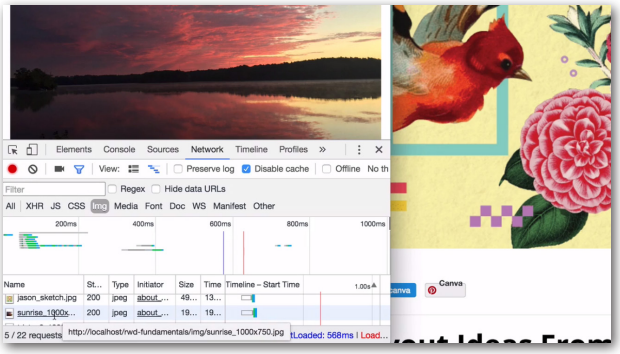
So the first one we're going to do is the Sunrise. This one, we're actually just going to swap out the same crop but with a little bit more variation. So we're going to stick with this image tag, but we're going to add a srcset in here and sizes as well.

So I'm going to break this onto a separate line so it's a little bit easier to look at. So we're adding sizes and we're going to add srcset. And we'll open up the final version so we can see what it's supposed to look like. And then we'll go through writing out the code.

So we've got the sizes. We're going to set our minimum width and the viewport unit references. So sizes equal and—now sizes is optional, but this gives you the ability to think through how you'll be using the image on that larger screen so that you can target something that is a little bit more appropriate for the eventual use of the image itself. It's not quite like sizing it for the container, but it's a little bit closer.

So I'm going to grab this string here and bring that back over. So we have our sizes, and we want to grab our srcset. So we've got four different images, and we have them listed in order from the 500-pixel-wide version to 780 to 1,000 to 1,500. And then we have our fallback image and our ALText.

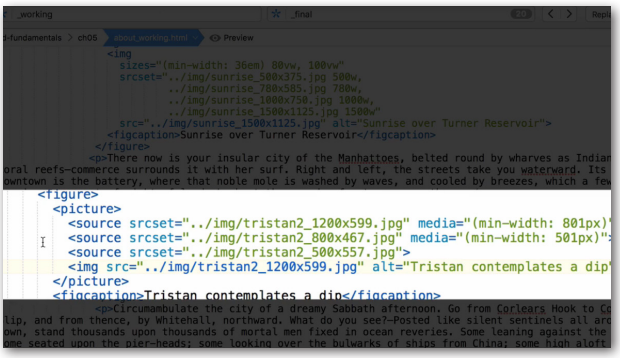
So if we save this and go back over to the browser, and if we go ahead and reload this page, I'm going to open up the Web Inspector and we'll just anchor this down on the bottom. And we'll just keep this network tab open, because this is where we can filter it down to see just what is being loaded with the images. So notice I've got just Image selected here.



Now I'm going to load the page. You can see we still have the 1,500-pixel-wide version being loaded. But if we make this window narrower and reload and look down here again, notice Sunrise 1,000. So we've seen that we definitely can get that smaller image to load. And if I pop this back out so we can go into the responsive view, toggle this button right here. And you'll notice that in this window, I've set 320 pixels wide. And using the Options I've added the device pixel ratio so we can pretend that this is not a retina screen.

So we've forced it to one, and if we reload the page—see, it's loaded that 500-pixel-wide version. So this is now loading a 60k image as opposed to, bringing this back out and loading again, down from 440. So we have an image that's perfectly sized for the device. It gives just enough clarity. It looks just as good on that small screen, but we have a savings of 380k, so it's going to be a massive difference in performance.

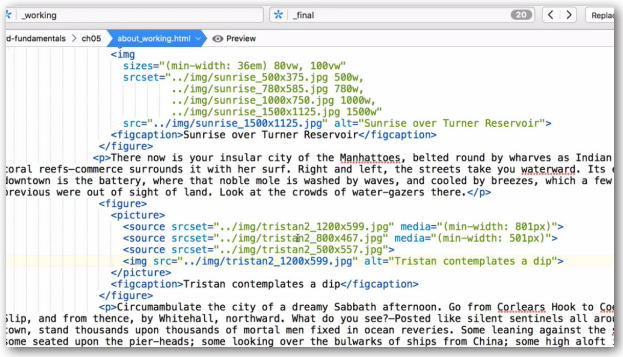
So let's take a look at what we can do with Tristan with this notion of art direction. So if we go back in here, we're going to be adding our picture element. And that image stays in as our fallback, but we're going to be adding in our sources above. Now if we go and look in this file here, it's these three lines. I'm going to copy these three lines and bring them in over here.



And see, it's similar to what we did above, except you'll notice it's actually in the opposite order. So where the browser above is looking through these files and finding the one that's most appropriate, and it is making the decisions here, in the picture element, it's going to go through this list of files and pick the first one that satisfies the requirements of the media

query. So rather than in CSS, where the last declaration wins, this goes through the list and finds the first one that matches.

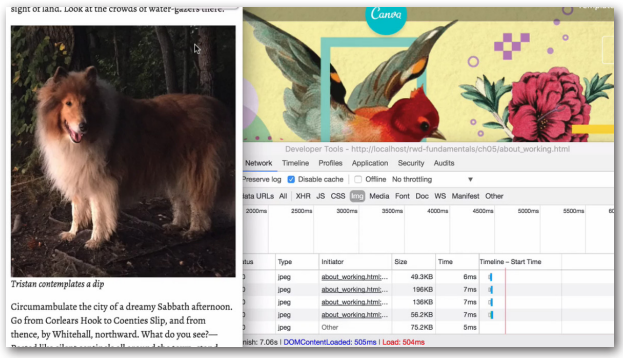
So if we look at the media queries, minimum width of 801 pixels, if that doesn't pass, it goes to the next one. If the window width is a minimum with of 501 but obviously less than 801, it's going to use this second image, else it's going to use the last one listed, which does not have a media query at all. And again, our fallback here is going to be this large one.



Now if we save this and go back over to the browser, when we're looking at this on a large screen, we shouldn't see any difference at all. We have that same familiar image. But now, as we bring this in, pay attention to the left-hand edge of this image. I'm going to make this window narrower.

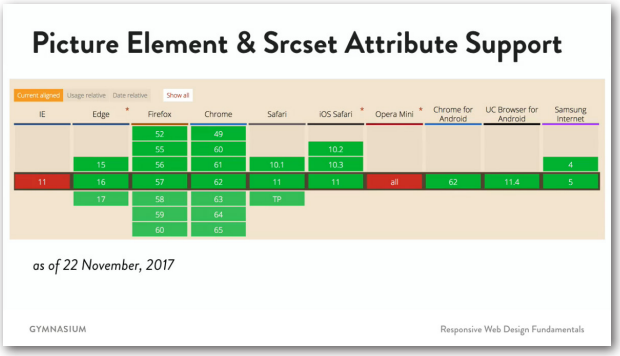
Notice how the crop of that image just changed. There's less water visible, and we've brought it in so Tristan stays a little bit more in focus. I'm going to back out again. I'm going to get that full image, and we look—the largest size, 1,200 by 599—is a 382k image. When we bring this in smaller and go back and look here, you'll notice when we load it's the 800-pixel-wide version down to 136k. And if we get down to the smallest screens, you notice we cropped it in again.

So here's our direction. We've kept Tristan the focus of this image as we get narrower and narrower. So on that small screen, where if we were to keep it at that original image, we would lose all that detail. We wouldn't really get to see what a handsome fellow he is. And if we look again, we've cropped this down to an even smaller image that's being loaded, the 500-pixel-wide version down to only 75k.



So we've dropped 300k here and about 380k in the other. We've saved almost 3/4 of a megabyte from the download on this page. And we've still given entirely appropriate images for that size device. And this is all really well-supported.

As we can see here from Can I Use, this is just a quick look at how well-supported the picture element and srcset attributes are across the spectrum of browsers. You can see that Edge, Firefox, Chrome, Safari, Opera, iOS, and Android are all supporting this really well. It's only the previous version 4 of Android and IE 11 and under, really any version of IE, don't have that native support. Neither does Opera Mini.



Now the good news is there's actually a polyfill that predates the spec. And the fantastic thing about this is going back to thinking through what was going on in the responsive image community group, some of the same people

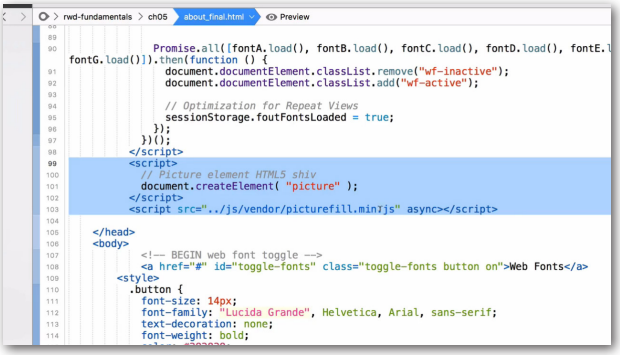
created Picturefill. Picturefill is a fantastic responsive image polyfill that evolved as the spec evolved. So it was one of those early iterations from the filament group that started to get us down this path of experimenting with better solutions for responsive images. As the spec evolved, so did this polyfill, and they got it around to the point where it's using exactly the same syntax.

So now what we can do is if we want to make sure that all browsers can make use of the same syntax in our page, we can add Picturefill. It will only be applied when absolutely necessary. So let's take a look and see what that looks like in code. We actually don't have to change the markup at all. We're just adding in the JavaScript.

So coming back to our working page, we know we're going to have to add some JavaScript. And if we take a quick peek in the root JavaScript folder in the Vendor folder, you'll see we've got Picturefill right here. So if we look in the about final, right below where we added this support for loading the web fonts, you'll see a few lines of JavaScript in here. It started on line 99, and it goes down to 103.

So including a comment, we're adding all of five lines. And you can see it's doing two things. It's adding in an empty picture element for older browsers that don't understand HTML5, and it's calling it's minimized version of the picture fill polyfill and loading it asynchronously.

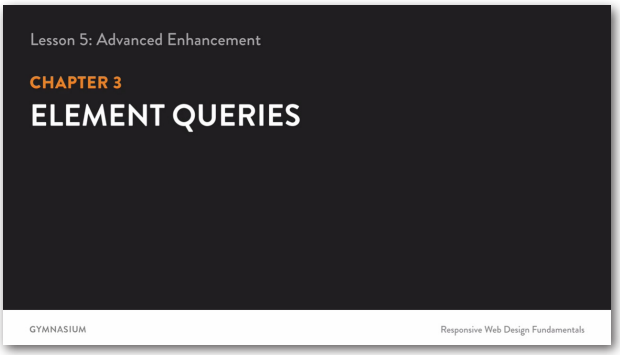
So we don't have to worry about holding up the loading of the page. But it will add that support without our having to change the syntax of the page at all. So we can simply copy this, bring it into our page, and go back up to the top here and add this in same spot, save that page.



We now have something that will work in virtually any browser we want it to without having to change a line of markup. So there it is. We've gone through and looked at the alternatives for responsive image formats. We've tried them out and added them and talked about the use cases of when you'd use Image versus Picture, and we've even added support for all the older browsers. So now we can go forth with confidence and know that our well-art-directed and optimally-loading pages are going to work great on just about any device and browser we can throw at them.

CHAPTER 3: ELEMENT QUERIES

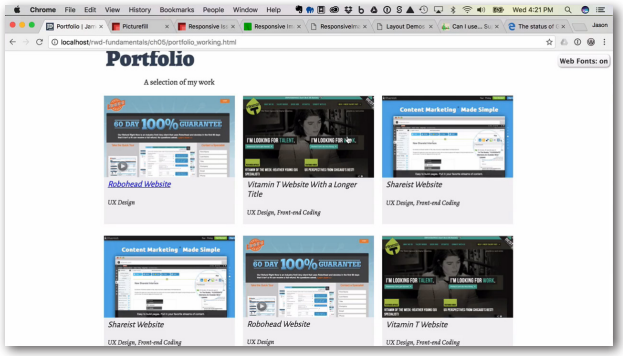
Well, container queries. So we want to talk about the limits of media queries and what is this new thing, this element or container query, and look at a way we might approach working with this, of course, with the disclaimer that it's not a spec yet, but it's an idea that's really important and a lot of people are looking for ways to solve this.



So let's pause for a moment and think about what media queries are actually doing for us. In the last chapter, we added these images and we used media queries and responsive images in order to have the

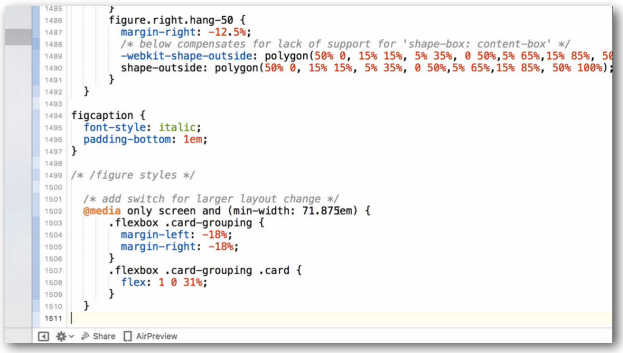
browser load the right image based on the size of the viewport or, in the case of a desktop browser, the size of the browser window itself. Now that's fine for most cases when you're using an image at a fairly large size that's going to be most of the width of the window, but what about on the portfolio page.

So if we go and we take a look at the top level of the portfolio—this is portfolio_working.html—we look at this card pattern. Now we've got relatively small images here, but what if they were a little bit bigger if we really wanted to optimize for this and make sure that even these images should be served appropriately for the size of the device upon which they're being viewed. But the way that they're being used is much narrower than the width of the window itself.



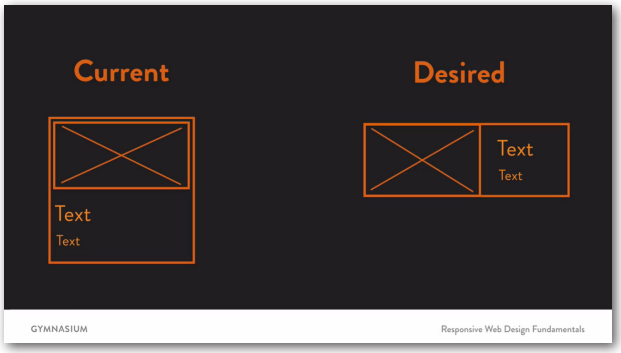
So an element or a container query is really about making the markup and CSS react to the size in which this element of content is as it's rendered on the screen. So in this case, we've got three across and there's some gutter on either side, so you can imagine that if we were to try and come up with a percentage, it's less than 25% of that window with, but we don't really know exactly how much. The notion of an element or container query is like a media query, but looking at the rendered size of that element of content itself. So think of these cards and we want to have the element query act on them based on the size in which they're placed on the screen. So let's take a look at some of the things we could do.

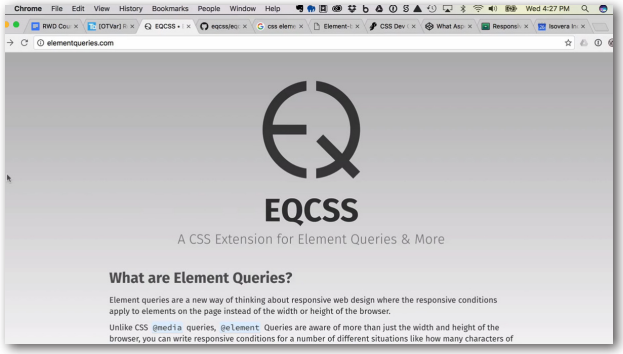
So I'm going to open up our portfolio working page, scroll down and find our little markup pattern for these cards right here. And we have a series of articles with a figure tag and a paragraph underneath, and this is all going to be rendered like so. And if we go and take a look at the CSS that we're using—I'm going to open up our working CSS file.



So before we start to take a look at the element query code itself, we're going to paste in, a little bit down at the bottom—so we'll look in our styles working CSS file, go all the way down to the bottom. And the first thing we're going to do is add in a media query for that card grouping that will address that wider screen and change that layout for those cards. So let's take a look and see what happens here. And we look at our layout.

So we added in this media query, and we extended the margins to the left and the right so that we now have the card stretching out to fill most of that layout, and then it drops back down once you reach the smaller media queries. But once it gets really big, you can see we're using much more of the screen. Now these images might not be laying out exactly the way we'd want. So instead of having these cards lay out what the image above, full width in the card and then the content below it, we might want to actually change that pattern so that we can have the image lay out half the width and have the text stacked next to it.



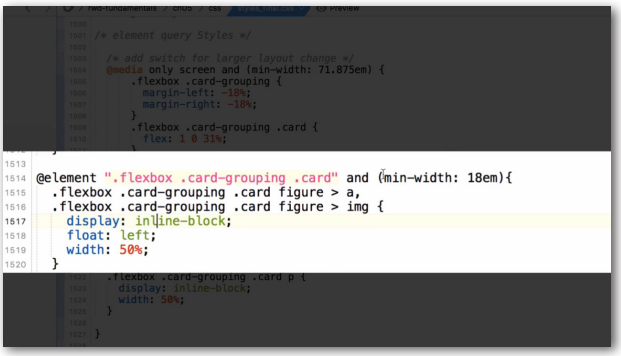


So enter EQCSS. It's a CSS extension created by Tom Hodgins to add an element query and some other stuff, but we're just going to use it to give us this notion of an element or a container query. It's really easy to add. From elementqueries.com, you can find it on GitHub and just download the entire repository. We're actually only going to use a couple of files. So if we take a look—I'm going to go back into our editor here.

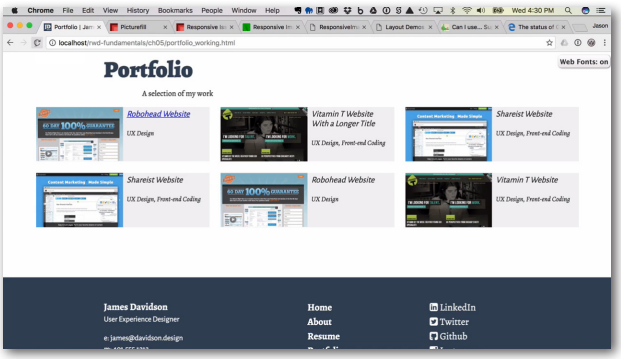
Look again in that root JavaScript folder in the Vendor folder and you'll see we've got EQCSS min JavaScript and a polyfill. So look at portfolio final, look all the way down at the bottom and you'll see that we've added in a few lines so that we can load both a polyfill for older browsers and the main one for the newer browsers to add the support for these element queries. So we're going to copy this and we're going to bring this into our working file. We bring that down to the bottom.

This is the only thing that changes on this page. We don't have to change the markup at all. We're just going to add some CSS. Now the element query syntax that Tom created looks really similar to our regular media queries. So I'm going to open up the final file. Now down at the bottom, same place where we added the other code, here's our element query. It's really, really simple.

So you can see @element instead of an @media, and you specify the selector. So in this case, we want to get down to the card level inside a card grouping when Flexbox is supported at a minimum of 18m. Now this minimum width refers to the element, remember, not the window itself. So it's only when this particular element is displayed at that minimum width, it will enact the CSS inside. And you can see for the image or if there's a link on it, it's going to display it as an inline block, float it left with the 50%, and all the content is going to also be set to 50% and displayed as an inline block.



So we bring this whole little bit with us, bring it over into our file, save it. And take a look at what happens when we load up our page. Look at that. So we didn't change a line of CSS, but we can determine that when it is presented wide enough, then we can change the layout and bring the text up next to it.



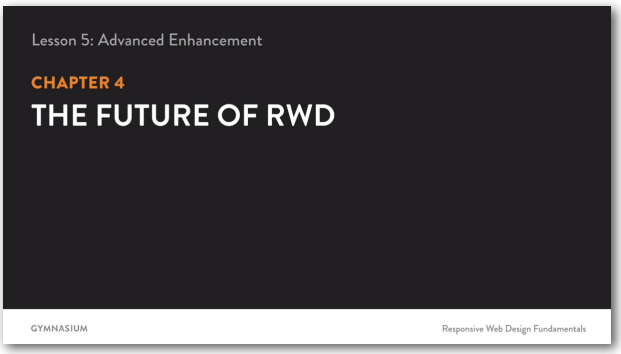
Now you could see that this is a pretty basic usage of it, but as soon as you start to play around with this, you can see that this makes your content patterns far more portable so that you can figure out exactly how this list of blog posts, or cards for your portfolio, or just a million other options, any kind of modular content, you could test it any size and have all of your markup totally self-contained. Think about pattern libraries and that sort of thing. It's such an incredibly useful addition.

So even though this isn't going to be exactly the same as the spec—the spec just hasn't been really fully determined yet—similar to Picturefill, I suspect that these kinds of libraries are going to be really useful and will evolve as things go. It's probably worth experimenting with it just a little bit just to see exactly how far we can push it, and see what we can do with our content and our design patterns and extend them to be used in more and more places.

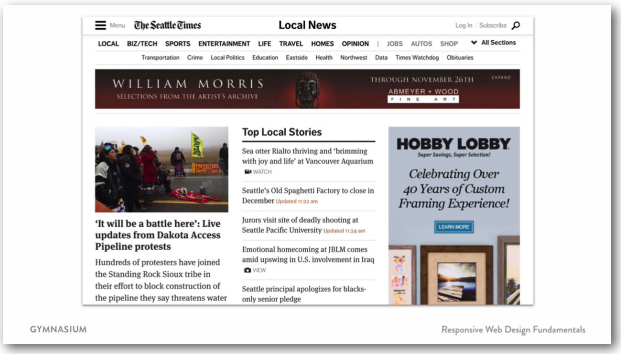
So we've talked a little bit about the limitations of media queries. They do get us a really long way towards much better designs on a much wider variety of screens, but it doesn't really fit every use case that we have, and extending it in some way with element or container queries is really going to be helpful in allowing us to take responsive design further and really think through some of the scenarios that we'll look at next. But at least, we've had a chance to try implementing it on our own site. And I'm sure that as you start to look at your own designs, you'll see places where they could come in handy as well. I hope you give it a try.

CHAPTER 4: THE FUTURE OF RESPONSIVE WEB DESIGN

Well, I may not be really able to predict the future, but what I can talk about are the limitations and frustrations that I think a lot of us are experiencing. Just now getting to master the basic ideas of making sites responsive, that's been hard enough. We really would like to make that part just a little bit easier, a little bit more routine, so we can start extending our design thinking and getting past the sort of basic squishy box approach.



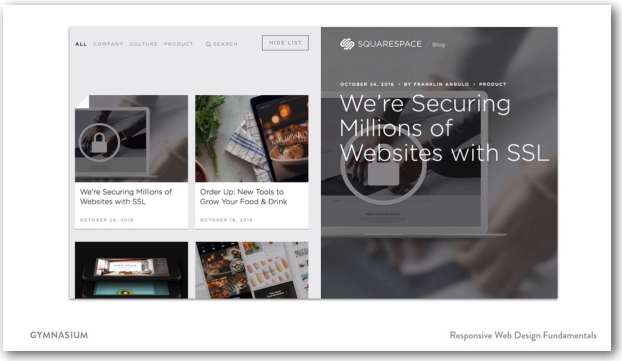
When we start to look at a lot of responsive sites, there are some great designs out there. The Mule Design team with the Seattle Times did a fantastic job with this website, but you see a lot of familiar patterns. You've got your basic card story on the left. And you've got smaller teasers coming down the center column.



A List Apart's a fantastic website, but you can see it's still a fairly basic grouping of content patterns. Sort of a two-third, one-third kind of a breakdown. You can guess pretty accurately what's going to happen with those topics once the screen gets a little smaller.

This little snippet from the Vox home page shows you another series of patterns, unfortunately, centered and completely indistinguishable as links. But you still have these same kind of patterns. You can guess, again, pretty accurately what's going to happen as the screen gets smaller. They're simply going to stack.

We've got the more visible cards on the Squarespace blog. This is a really pretty nice layout, the way this comes out. But it is still that really familiar pattern of image, text, boxy kind of layout. Again, there's no mystery to what's going to happen as that reflows.



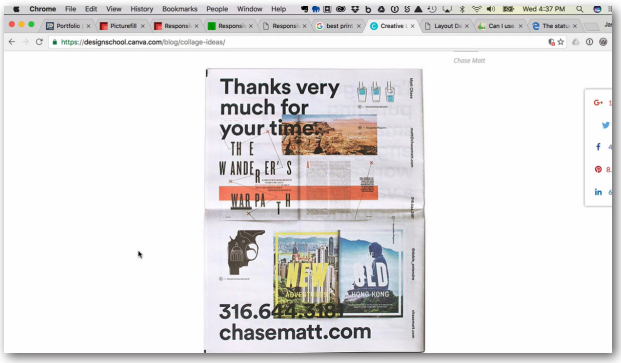
And even a fantastic site like STAT News, they've taken it a little bit further by adding this notion of something being in a stack. But it is still little boxes that line up and then rearrange as you move through the different break points.

I've been a graphic designer for a long time. There are a lot of really beautiful layouts out there, in books and publication design, and newspaper design. This stuff has existed for decades. And the problem with looking at this stuff through the lens of the web is that it's been so difficult to create designs that work with all the technical constraints, that we haven't really been able to get past that, to really start thinking about what are some fantastic layout ideas.

So I just Googled, best print layouts, and found a really great page on the Canvas site. So if we take a look at this page, now, the layout itself here isn't rocket science. But the kinds of things that they're showing off kind of are. So this cover from Dash, you can see the way the text and image really react to each other.

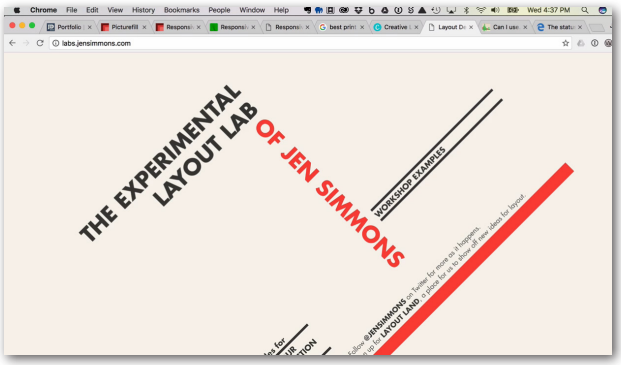
Some other print layouts that we see. You can start to get a sense of boxes of content. But they're really using and reacting to the page size beautifully. And we keep going down. There's so many ideas in how we might structure text and columns, use the entire layout, stretch things, move them around, reorder them. There's so many great ideas here that we could be doing on the web, but we've had such a hard time just kidding the basics going that we haven't really been able to extend beyond.

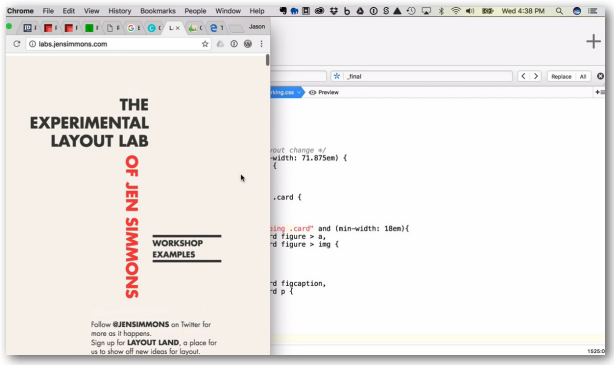
And we haven't had as many of the CSS layout techniques. So when you think about grid, and think about Flexbox, and viewport width, and viewport height, and look at a layout like this one, and think we really could do an awful lot to position and layer content in a really fascinating way. We could recreate all of these techniques and make use of all the real estate that we've got.



Now I'm going to come back for a second to Jen Simmons' lab site. Now this is doing a fantastic recreation of the cover of a book on Bauhaus book design that I actually have at home. So look what happens with this layout at different screen sizes. So we bring this in a little bit more, all of a sudden, layout changes. Still using lots of really great ideas. Really interesting tension that she's creating. And we get narrower still, it's reflowing yet again, and again.

So Jen's really thought about how can we take the ideas that are presented in these layouts and adapt it to be something appropriate for the size of the screen. And really done a fantastic job in reinterpreting those design ideas into something appropriate for the web.





We really need to start thinking this way, as we move forward and get past our squishy boxes, to really use the whole screen. Because that's where things are going to get really exciting. It's not just what we do on small screens, but what about these 27-inch iMacs and super high resolution screens that we're all toting around? There's really just a whole world that we haven't explored yet, and we now have the tools to do it.

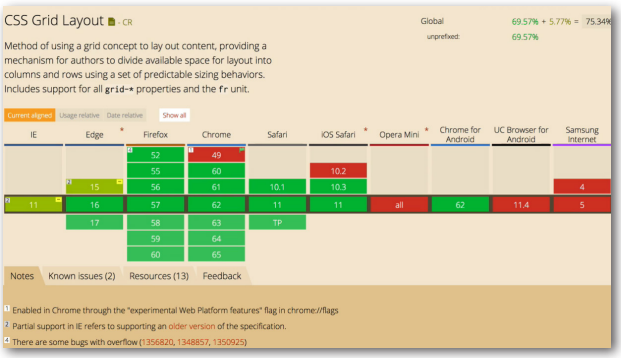
I hope you'll all join me and push forward together, and really start to extend our thinking, and bring a little bit more interest, and a little bit more variety in the way we lay things out on the screen.

So the layout we just saw from Jen Simmons is technically feasible due to CSS grid layout. Another name you should familiarize yourself with when it comes to grid layout is Rachel Andrews. Rachel has been doing an amazing job shepherding the CSS grid spec for a number of years, and you can see some of her work and benefit from her experience by checking out her site GridByExample.com, or following her on Twitter.

What's happening with the CSS grid layout and browser support is historic. In February of 2017 0% of browsers supported that feature. And within two months, there was approximately 80% support. As I'm recording this in the fall of 2017, let's take a look at the state of support on Can I Use. It shows us lots of green. And green is good. It means that the feature is either fully or partially supported.

And we have full support in Firefox, Chrome, Safari, iOS, and Android. And we have partial support on Microsoft IE 11, And Edge 15, with full support in Edge 16, due to release in October 2017.

So by the time you're watching this, it's most likely going to be supported in every major browser that shipping. Amazing. Between this, new developments in font technologies, and formats that are coming along, we've got so many new tools at our disposal to really extend our layout vocabulary.



So we've reviewed the current state of the responsive web and some of the limitations that we have, and some of the new ideas that we can start to play around with. And we've also looked at other directions, and techniques, and the status of them out in the wild.

Most of these things we can start to play with today, some of which we can actually incorporate into our production designs using standard techniques for progressive enhancement. Others we're simply going to have to wait a little bit, but starting to get to know them now is going to really pay dividends in our level of comfort, and our ability to extend our thinking going forward to create some really fantastic new designs.

So next up, we'll be talking starting to tackle performance, and make sure that we've done all we can to take our portfolio site and make sure it's working not only in an appropriate manner for the size screen upon which it's being viewed, but also being served as speedily as we possibly can.