GYMNASIUM

RESPONSIVE WEB DESIGN FUNDAMENTALS

Lesson 2 Handout

Basic Mechanics and Leveling up

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. For many years, web developers and designers used fixed-width pixel measurements as the foundation for page layouts, primarily because desktop screens were how people viewed the web.
2. When mobile arrived, as well as tablets and other devices, the fixed-width layout became a problem and the industry has slowly shifted toward using flexible layouts, which allows the content to scale based on the size of the user's screen or device.
3. Defining the maximum width for your content is a powerful way to improve the appearance of your page layout, for example to limit the line length of a paragraph in order to improve readability. This is typically achieved with the CSS *max-width* property.
4. One technique for converting pixel values into percentage values is to use this formula: Target divided by context equals value.
5. Responsive video presents a unique set of challenges when you want your video to scale based on the size of the screen. The technique used in this course is to rely on the Fitvids.js jQuery plugin.
6. CSS Media queries use specific code syntax (typically the *@media* rule) to help you adapt your design to the resolution of the viewport, i.e. the user's screen or device. With media queries you are able to load specific styles in the user's browser when specific conditions are met, for example the size of the screen they are using.
7. Two specific media queries are extremely common in responsive web design: *min-width* and *max-width*. When using min-width you are essentially saying, only apply specific styles when the browser is **greater than** a certain width. When using max-width you are essentially saying, only apply specific styles when the browser is **less than** a certain width.
8. A good rule of thumb for responsive design is to only use pixel measurements for border-size, box-shadow or text-shadow. Otherwise, use a relative size of measurement such as the em in order to make sure your content scales appropriately across any device.

ASSIGNMENTS

- Quiz
- The second assignment is designed to help you further your understanding of media queries. Go to the following site: http://bradfrost.com/demo/ish/
    - This is a viewport resizing tool developed by Brad Frost called Ish. First, read the details of what the tool does on the site. Next, in the top-left corner, type in or paste in the address of a website you would like to test. This could be your own portfolio, or, it could be any responsive website of your choosing.
    - The objective is to identify and understand the existing breakpoints on the website. If it is your own site, begin looking for content that you want to fix, if you are using someone else's site, look for content you would fix as the designer.
    - Questions to ask yourself— How many breakpoints can you count? Are there places where the line length makes it hard to read? In the layout, are there any surprising or jarring content changes?
    - Take note of all your responses and then go to the classroom site and make a post for lesson 2 with your observations. Be sure to include the URL of the site you visited. Additionally, find another student's assignment and give them feedback as well.

INTRODUCTION

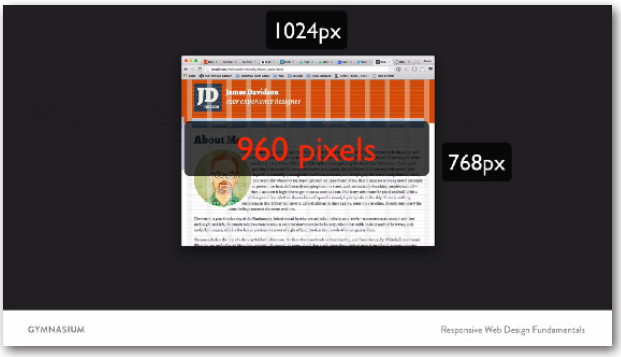
(Note: This is an edited transcript of the Responsive Web Design Fundamentals lecture videos. Some students work better with written material than by watching videos alone, so we're offering this to you as an optional, helpful resource. Some elements of the instruction, like live coding, can't be recreated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

OK, Lesson 2, basic mechanics and leveling up. We're going to go through converting a few things in this lesson. We're going to get started on making our layouts move from fixed to fluid and start making our pages a little squishy. We're going to look at flexible media to get our pictures and video scaling as well. And then we're going to move into media queries to let that design react and reflow based on screen size. Finally, we're going to talk a little bit about units of measure and when a pixel is not really quite a pixel.

CHAPTER 1: FLUID LAYOUTS

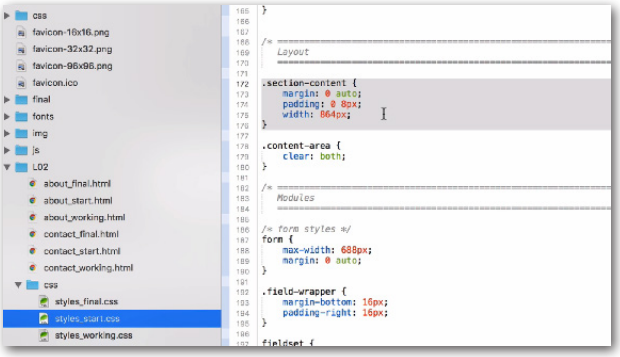
What you'll learn here is a little bit of history. We're going to talk about screen size norms and other kinds of fantasy fiction that we've experienced over the years. We're going to review our mockup and see what kinds of things we should change. We're going to convert that layout from fixed pixel widths to percentages.

First, there was this. 640 by 480—that pixel measurement was something that was basically a standard in the world when the web was born and it was quite a while before that started to change. It did eventually move to 800 by 600. That gave us a little bit more room to work with. And then we pretty much decided that when we reached 1024 by 768, that's where the world was ending and we could standardize on a 960 pixel grid. And things stayed that way for quite some time even as monitors got bigger and bigger, and we realized that even as the resolution went up, our content was staying really tiny right in the middle of the screen.



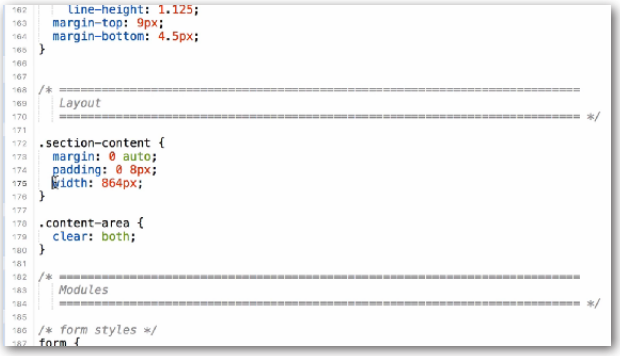
Then this happened. Thanks to Brad Frost for supplying the photo. We can see there are—just in this photo alone—dozens of different screen sizes and browsers that we have to contend with, and it's quite clear that one size of content is not going to work. So here's where we get to see our existing layout. It's pretty static and you can see it doesn't scale. So if we want to make this scale, then we need to start thinking about what elements need to be able to move from a fixed width to a percentage.

So let's start out looking at section content and main nav. Now a quick note, you'll see there's a starting file, a working file, and an end file for each page in the site, including the style sheets. So if you take a look in the files that we have for the project, you'll see the starting file in section content, that the width is fixed at 864 pixels and the main nav is the same. And what we want to do is start to change that around a little bit.



We want something more like this, maximum width, so that we can keep the layout looking like we've designed it at the largest screen size, and then a fluid width below that of 90% so as browser window gets narrower, that it starts to scale with the screen. So if you go back to your files, you might want to start out with the working file and make changes in there. So you can always go back to the starting file if you need to reset and start over.

So I'm going to go ahead and open up the index page and we'll open up the working file. I'll leave the start one alone. I'm going to do the same thing with the working CSS file. Now we want to scroll down until we find the layout section, and that's going to be—there we go—starting on line 168, if you have line numbering on in your code editor. You can see that we have a width of 864 pixels and we want to change this to actually be the max width. Then add a line and set a width of 90%. So what this will ensure is that as the page gets smaller, the content will scale to 90% of the window width. And thanks to our margin of auto, it will stay centered, but it will cap at a maximum width of 864 on a larger screen size.

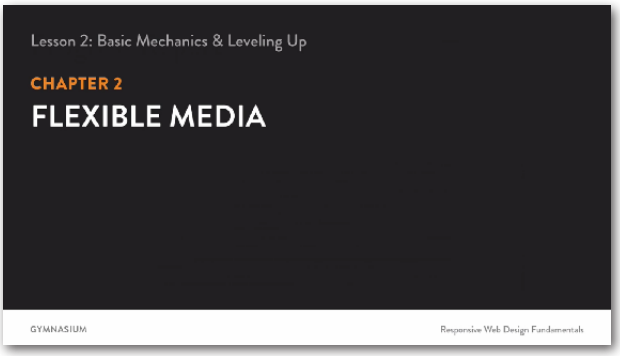


We also need to do this for the main navigation. So if we scroll down a little bit further, it's down here. And the line we're looking for is around 362. I'm going to do the same thing. I'm going to change this to a max width, add a width of 90%. OK, I'm going to flip over to the browser and we look at the index working page. I'm going to go to the About page. It's a little bit easier for us to work with. There's fewer elements on it. And if we take our browser window here and bring this in, we can see that the main nav and the content is now scaling. There's a few things that we'll need to change, but we're on our way.

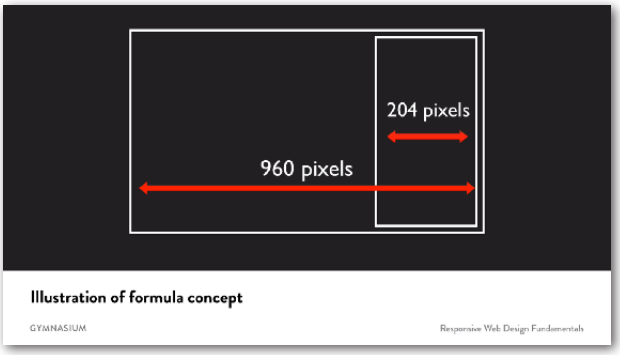
One thing we might want to consider is that maximum width. Right now on this browser, it seems fine. It's a reasonable line length for our copy, but it might be that on a larger screen, we might want to change that maximum width to something even wider and take advantage of that larger screen. Now this is something you can experiment with on your own system and see at what point does the text become unreadable because the line lengths get too long and the layout just doesn't support that screen size.

Then you have a choice to either change the layout or simply cap it and let the contents stay centered. That's the choice that we've made just to keep this a little bit simple so we can keep moving along. Now we've done two simple things to get us on our way. We've converted layout from a fixed width to fluid, and we've added a max width just for now to match that previous fixed design.

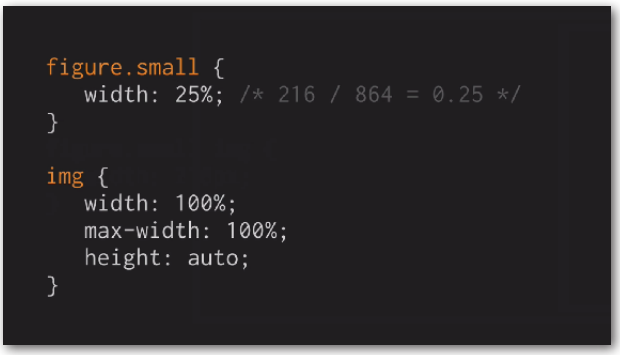
CHAPTER 2: FLEXIBLE MEDIA



Now we're going to start to think in containers, and the reason we need to do that is to allow those images to scale best and let the browsers handle the work for us. We're then going to think about a little bit of math. Ethan Marcotte gave us a sneaky little formula that we can use that I promise is not that hard. We're going to convert from pixels to percentages for widths, and we're going to just let the browser take care of the hard stuff. Finally, we're going to dress things up a little bit. We're going to add some little CSS bling to our layout, and we're also going to add in some video.



We're going to start with the math. I promise it's not that bad. Target divided by context equals value. It's a really simple formula and it works just about everywhere. So you can see in this illustration, we have a content area that's 960 pixels wide and a sidebar area that's 204 pixels wide. Now if we want to figure out what the percentage for that sidebar is of the containing element, that 960 pixels, we have a handy formula—target divided by context equals the value we need. So if our target is 204, our context is 960. 204 divided by 960 is 21.25. And therefore, our percentage is 21.25%.



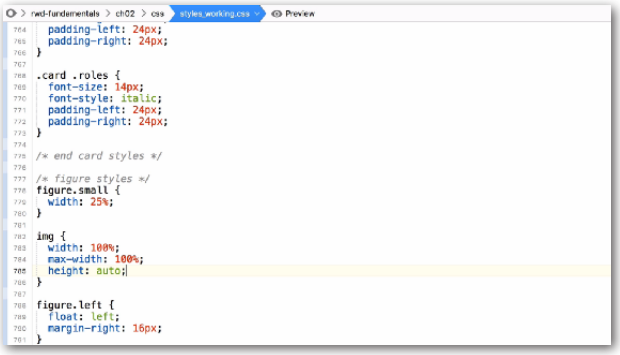
```
figure.small {
   width: 25%; /* 216 / 864 = 0.25 */
}

img {
   width: 100%;
   max-width: 100%;
   height: auto;
}
```

So we need to go from specifying the image size in our CSS, where the image is 216 pixels wide, to actually sizing the container, the figure itself, to width of 25%. And we see a formula, 216 divided by 864 is 0.25 or 25%. Our image, we want to fill the space. And the reason we do this is to get browsers to properly scale the image and give it the best smoothing effect.

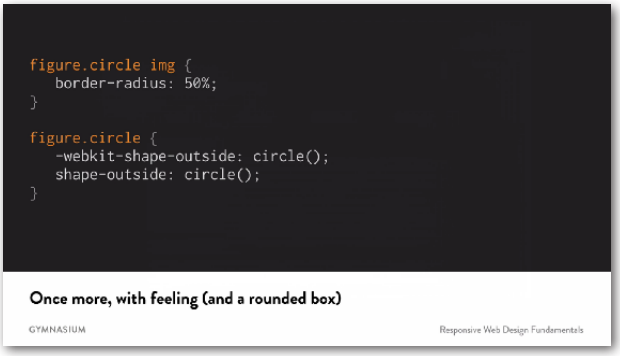
We want to set the image to 100%—a maximum width of 100% so we know that it always fills its container and set the height to auto. The reason we set the height to auto is, on older browsers, it may scale it disproportionally, leave the height set to what it should be in its natural state and then only scale the width. We want to keep the images looking good and don't get into any weird distortion issues. So we just put this in here and make sure we cover all our bases.

So going back to our code, were going to look around and find the containers for the figure. That's down in the bottom of the file. So if we scroll down all the way towards the bottom, we'll get to the section right at the end—right around line 777. And we have the figure styles. You'll see that it's set here. The size is on the image itself, and we need to change that to be a width that's actually on the container.

So we're going to do a couple of things. We're going to remove this and we're going to convert over to our formula. We know 216 into the containing size is going to be 25%. So we're going to set the container to be 25%. And we're now going to make sure the image scales properly, and we're going to set the width 100%—max width of 100%, height, auto. There we go. So now if we go back over to our page and we hit refresh, and we start to scale again, you'll see the images are now scaling cleanly with the design itself.

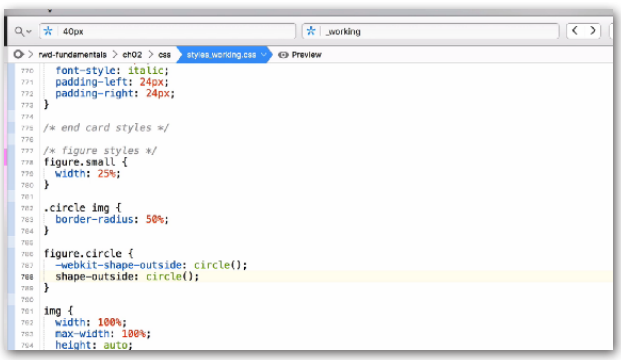


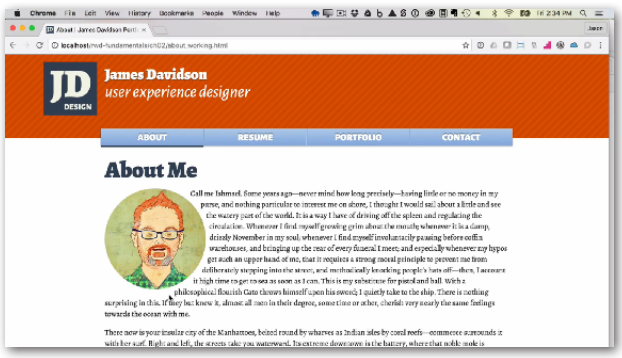
Now we want to dress it up a little bit because the square images are just, well, a little square. So we're going to add a border radius on the image itself. And then on the circle, we're going to take advantage of a shape that's something new in CSS3. You can see we still have to supply the vendor prefixing for webkit, but the standard CSS syntax is here and should work in every other browser that's currently shipping. So this is going to give us around image, but also a round space in which it sits.



So if we look in our HTML and scroll down until we find that first photo, you'll notice that there are a few extra classes on here that are going to be helpful for us to tie into. We know this figure is aligned to the left. We know now that we want it to be a circle and that it should be the smaller size. So we're going to tie into all of those things when we look at our CSS again. And see here that we want to add this border radius, we need to do a couple of things.

For an image that has the circle class and the image inside, we want to set the border radius of 50%. That will ensure that it gives us a nice clean circle all the way around. And on the figure itself, when it has the circle class applied, we're going to add in the shape outside.



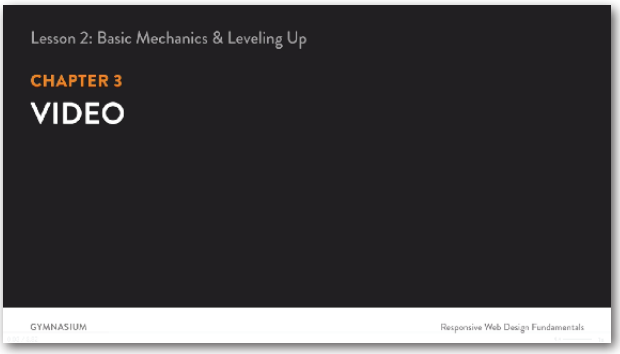


We save this. Now let's go back and take a look at our page and hit refresh. Pretty cool—we've got a circular image, with border radius, and the shape outside gives us this nice clean wrap around the image itself. It's a really nice effect and adds a nice level of polish instead of typical boxy cutout we normally see. Now one note about those utility classes—if we go back to our CSS and look at the HTML page, again, we've got left hang-50—which we'll get to in a little bit—circle, and small. The reason we use these is so we can layer in the effects that we need with a minimal amount of additional CSS.

So looking here, we only set the border radius when we've put the circle class on the containing element for the image, and we only set the shape outside on the figure itself. Keeps things nice and clean. The nice thing about both of these is that if the browser doesn't support it, it just falls back to a simple square image. We've not lost anything, but we've added something through progressive enhancement to enhance that experience for browsers that can support it. And I'm happy to report that every shipping browser right now is going to see this effect.

CHAPTER 3: RESPONSIVE VIDEO

FitVids.js is a little library that was put together by Dave Rupert that does a really fantastic job of helping video that you embed from YouTube and lots of other sources scale along with the rest of your design. Now it is a jQuery plug-in, so it does require that you have jQuery loading in the page, but it's really simple to implement. You have to make sure you have jQuery, then we add the jQuery FitVids' JavaScript, and add this one function to our main JavaScript file. And what we can do is instead of identifying it with an ID, like it is in this example, we can identify it with a class and it will work with anything that we put on the page.




```
<script src="path/to/jquery.min.js"></script>
<script src="path/to/jquery.fitvids.js"></script>
<script>
  $(document).ready(function(){
    // Target your .container, .wrapper, .post, etc.
    $("#thing-with-videos").fitVids();
  });
</script>
```

So we're going to grab a few things from our final files and change around a couple of things that we're referencing here. So if we take a look at the about_final page, scroll down to the bottom of the content, and the first thing we're going to do is grab this line here, which is a video that I grabbed from YouTube. I Copy that, and I'm going to bring it into our working file. And before we do anything else, let's take a look, reload the page, and you'll see we've got a nice trailer for Lassie from 1994. It's my little tribute to Tristan. Kind of a sad thing, not scaling as nicely with the rest of our stuff. So let's fix that.

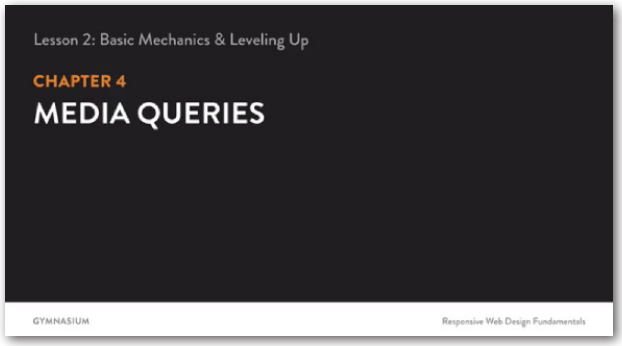
If you look in the vendor folder, you'll see jquery.fitvids.js. We want to add a reference to that file on our page and start to make use of it. So if we look down here at the bottom, we're going to add in a line after we load jQuery—script source equals—and we want to go back out to that route JavaScript folder in the vendor's folder—jquery.fitvids.js. Close out our script file. So we have that loading, but we still need to go into the JavaScript for this project.

And you'll see, I've added the function in here for us, and I've specified this video container. Now let's go back and take a look at that working file again where we placed it in the HTML, and you'll notice I've put this div wrapper. That's all you've got to do. If you have your embed code, this is straight from YouTube, the iframe. I'm not taking the width and height out. I'm just leaving it there, wrapping it with this video container, and hit Save. I'm going to go refresh the page again. And there we go. Really easy, don't have to do anything other than drop the embed code in, make sure it's wrapped with a div with that class on it, and FitVids takes care of the rest. It's really widely supported as long as you've got jQuery.

So there we have it. We've converted all the images from a fixed size to percentages. We've added the bling, we have a little transition, we've added some border radius, we've added shapes. We've added FitVids to embed a video, and we have it all scaling just like we don't care. All right, so let's move on.

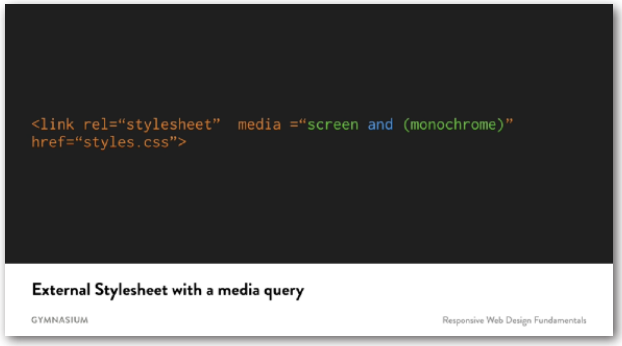
CHAPTER 4: MEDIA QUERIES

Now, we're going to get to media queries. So what we're going to learn, we're going to look at minimum versus maximum width media queries. We're going to talk a little bit about what they are in general. We're going to we talk about units of measure and why pixels are going to get you in trouble. And we're also going to talk about the notion or term breakpoint and how that relates to larger issues of layout versus the notion of tweakpoints, a phrase coined by Jeremy Keith, to address small issues in layout or design that don't really require major changes.



So what's a media query? A media query is something that in your CSS helps adapt your design to the resolution of the viewport in which it's experienced. I'm not going to get into a full description of how media queries work in this lesson. If you've never seen a media query before, if you need a refresher, in Aaron Gustafson's course, Modern Web Design here on Gymnasium, he does a thorough job of explaining the concepts in lesson 5 chapter 10, called, appropriately enough, Media Queries.

Having said that, here's a quick refresher of media query syntax. Let's start with the basic link to an external stylesheet—link rel="stylesheet" href="styles.css. Now, let's add a media query with two conditions included. So we'd have link rel="stylesheet" and media screen and monochrome href="styles.css.



In this example, the stylesheet will only be loaded if the two conditions are met—condition one, the user is using a device with a screen, and condition 2, that screen is

monochrome. So this stylesheet might be useful if you want to optimize your page for a device, such as an ebook reader, for example.

Now, let's look at media queries using a different set of conditions and using a different method. In the previous example, we were using the media query to load an external stylesheet. But in this example, we're putting our media query directly into our existing styles using them @media rule like so—@media parentheses max=width 1,000 pixels close parentheses, and then your style's color, red.

In this example, we're using the prefix max, which means we can think of it as a maximum constraint. So the styles will only be applied if your browser's viewport is equal to or less than 1,000 pixels. There's also a prefix min, which you can think of as a minimum constraint. So if we use this syntax, @media min=width 1,000 pixels color red, then the styles will only be applied if your browser's viewport is equal to or greater than 1,000 pixels.

Here's a little cheat sheet to help you remember the difference between the two. min-width can be interpreted as greater than a certain width. And max can be interpreted as less than in certain widths.

My general course is to always go with minimum width. It's less confusing to settle on one way. And it means less processing for small screen devices. This is important for a number of reasons.

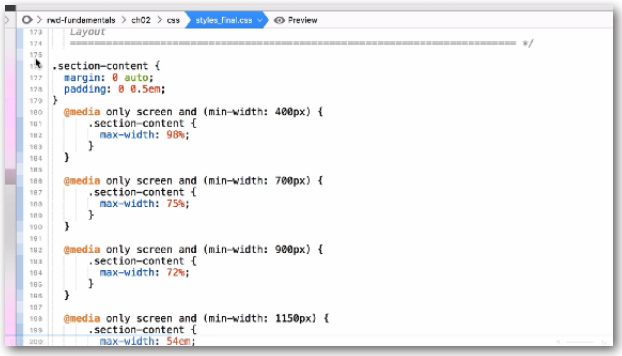
For one thing, it's all simpler for you conceptually. If you always have your media queries set to work small screen and then when it reaches a minimum width, you change the layout a little bit, and you add a little more CSS. The bigger issue is that there are a lot of devices out there that still either don't support media queries when they access the web, or there are issues with latency.

Think several billion cell phones—they're all going to have some amount of speed penalty. And the last thing you want to do is force them to process more stuff. By starting with minimum width media queries, you always ensure that the least amount of CSS has to be processed in order to render the page on a phone.

So let's take a look at our file. And we're going to add some adjustments in layout for both the maximum width of the overall layout, as well as dealing with some things in the footer that aren't looking quite like they should. In a CSS, we are going to cheat just a little bit, and we're going to open up the final styles file and scroll down and look for our section content. And again, it's around line 176 or so.

And we're going to grab this chunk, bring that over to our file. And we'll talk about what that's actually doing. So when we come down here to our layout section, I'm going to paste this in.

So what's going on here? We have a series of media queries that are building up from a range of different widths. So as soon as it reaches a minimum width of 400 pixels, 700, 900, 1,150, or 1,300, and setting a maximum width on that design.



And you'll notice that we're topping out at the second largest one here at our expected 864. And we've added another one for a slightly larger screen. That will address a slightly wider content area.

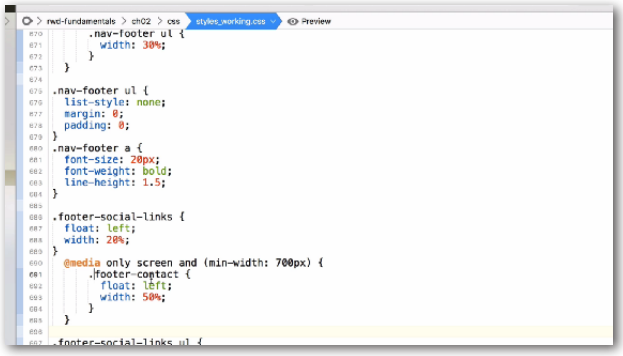
So for our minimum width, for this base selector before we add the media query, we want to just simply remove these two bits right here. And as soon as we do that, and we need to do the same thing for our menu. So we copy this and go down until we find that main menu is right here. I'm going to paste this in, update the selector—main-nav, do that again for each one of these.

So we want this to match the section content according to our layout. And remember to take these two lines out here. Save it. And let's go take a look. Reload our page. And as this scales down, you can see that our maximum width changes, just the way we'd expect.

Now, one place we still have a little bit of trouble is in the footer. You can see, as we scale this down, it gets a little bit jumbled down there. So we probably need to add a media query in there as well. But we probably only need one. So you can see, as soon as you get out to roughly this sort of larger tablet size, it's actually pretty good.

So let's go back, and we're going to look down for our footer. We're going to grab this media query and take it with us. And we need to scroll down until we reach the footer styles, around line 625 now. And if we look for that contact block, these are the ones that we converted to percentages a little while back. And we're going to add a media query in.

Now, what we want to do is think about what do we do with this block. So the contact block, let's have that one go full width. So we're going to update our selector—footer-contact.



And when the screen reaches that minimum width, we're going to float left and then assign the width of 50%. But at its smallest size, we actually just want it to occupy the full width. So we'll add this.

Now, we're going to add something similar to the other two blocks. It's our nav-footer. And again, these two can stay side by side—the nav-footer foot or in the social links. So we won't worry about changing the float.

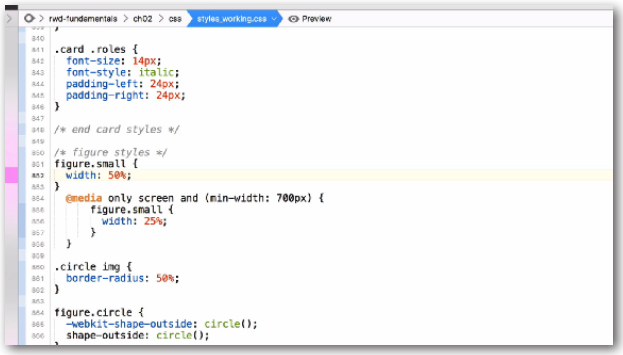
But we do need to make sure, at its smallest size, it's going to be 50/50 and that it changes to only 30% and 20% for the social links. And the float will still be in place. We're only changing what we need to change—20% at the larger screen size, 50% at the small one.

And let's go back and take a look. I'm going to bring this down, scroll, and refresh our page. Look at that. Perfect. So we have these two blocks line up side by side and the address coming across.

We think we might want to do something with those circular images. So we want to react with grace, and so the images can scale along with the rest of the layout. So we're going to go and take a look in our CSS again for the figure element, which is down at the bottom.

And we can see, there are a couple of things that we can do. We can add a media query in. So again, at roughly that 700 pixel size, we're going—whoops—for the figure with that small utility class on it, we'll set the width to 25%. Don't have to worry about the float. And for the smaller screen sizes, we'll set it at 50%.

As soon as we go and refresh that, we'll scale this down and see that on small screens preserves the nice wraparound with that shape outside. But the images look a little bit nicer on that small screen. So let's come back out here and think about one more little refinement.



Now, this is one that's really mainly just for us. You might have noticed here, we've added a transition. And that transition really just helps ease that scaling from one screen size to the next. It's not something that you'd see unless you were actually scaling the window and watching for it. But there's not really a cost here in adding this kind of transition.

So we can go ahead and paste this text in. And I've included some vendor prefixings to make sure that it works in all places. And if we scale this, refresh our page, you'll see that it just adds a nice little transition effect as people move the browser around. And really, that may be mainly for us, us silly people that actually resize the browser window whenever we visit a website, but it's a nice touch.

```
figure.small {
   width: 50%;
   -webkit-transition: all linear 0.2s;
   -moz-transition: all linear 0.2s;
   transition: all linear 0.2s;
}

@media only screen and (min-width: 700px) {
   figure.small {
      width: 25%;
   }
}
```

Now, there's one other thing as we scale this around that we notice it is a bit of a problem. And that's that disappearing logo. So here's where we might think about that notion of a tweakpoint. We want to do something to make sure that logo stays visible, but we do like this overhang effect.

So let's take a look and see what we can do here. I'm going to scroll back up and look for the logo div. And it's going to be back up here towards the top of the file right around line 325 or so. And we know we need a media query in here.

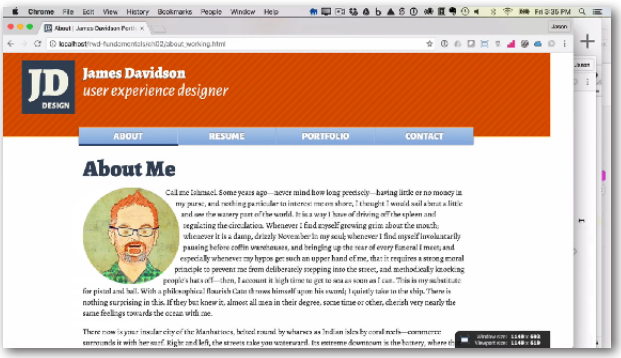
So first off, I'm going to go back up to the top of the file and grab my sample media query. And I'll go back down to the logo. Now, let's add this in and think about what we'd be doing here once we reach that 700 pixel width.

We're going to set the margin-left, so minus 128 pixels. And when we're at the smaller screen size, we're actually going to just not have any negative margin at all. So when we refresh the page, it should look the same here. But when we get to a smaller screen size, we'll see it comes back in, and we don't lose anything.

But we do have this awkward little moment here where we lose it. And that's immediately after we reach that first breakpoint. So let's add a tweak.

So if we know that it's just a little bit off screen, let's see what happens if instead let's go when the screen is a little bit wider. And at that 700 pixel range, we'll only indent it perhaps half as much. Let's try 64, Save and refresh.

Again, move back down, it's pretty close. So you can see here that it jumps back out. And that's probably right where we hit that second media query that we just created. So maybe we need to increase the number where that kicks in.



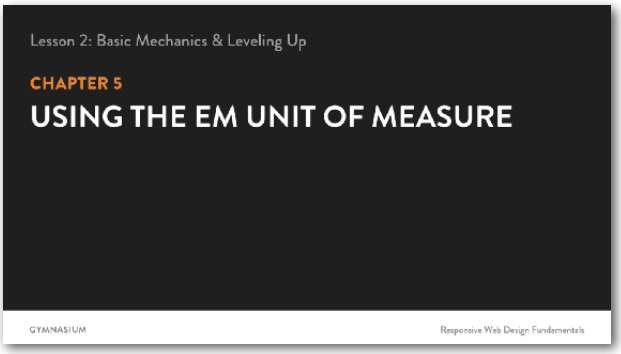
So let's try that and make the window a little wider. So it's still just a little bit off the screen. So why don't we round this up to 900 pixels and refresh.

And as we scale out, there we go. So had to add a couple little tweaks in there. But we have our images scaling well and the placement of that logo looking just the way we'd like it.

We've added breakpoints to address a range of screen sizes. We've added media queries for our overall layout. We've added media queries for figures and the head elements wherever they're needed. And we added a couple little tweaks to get that layout working just the way we'd like across that whole range of screen sizes.

CHAPTER 5: USING THE EM UNIT OF MEASURE

So let's close out the lesson by talking about units of measure. Bonus points, if you know what album I'm referring to.



Unless you're a border size, a box shadow, a text shadow, you should not be set in pixels. That's the basic rule of thumb for a responsive design. So we want to make sure that we are tied to something relative, that will scale appropriately across any device.

So we're going to convert any remaining pixel measurements to EMs—we're going to talk about what EMs are. And we're also going to be sure that we do this in our media queries, as well, and we'll talk about why in just a moment.

The main units of measure that we're going to talk about our EMs, REMs, and how they compare with percentages. There are some new units in town—viewport width viewport height. They're interesting, and

they're great for layout, and we'll talk about how you can use them with type, and why you may or may not want to.

Well, they're fake. Really from the very beginning, they've been measured differently in that pixels per inch on Mac and Windows. Started out at 72 and 96, and that difference has only grown as screen resolutions have gotten better and better.

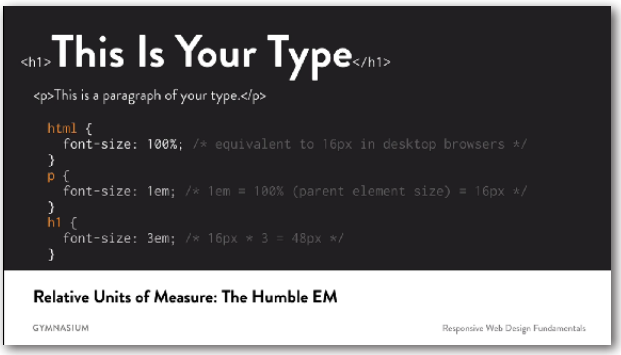
We've now fallen back on something called a reference pixel dimension, that does not equal your screen resolution. Which is really complicated so when people talk about retina displays or other kinds of high pixel density displays that you find on the modern iPhones and Android devices, quite often, it's as many as 400 pixels per inch.

Type set at 100% on desktop has been equivalent to 16 pixels for years. On phone devices, it's a little bit different, but the advantage is phone device manufacturers do a lot of testing and they always have something readable at 100%. So it's generally close enough for us to use that as a basis for all of our scale.

An EM is a measurement that's equivalent to one unit of the current size of type. So you can generally rely on 100% type being equivalent to 16 pixels and therefore, one EM is also equivalent to 16 pixels, until you start changing that EM value or the base pixel size.

This is where people get into a little bit of trouble with EMs. We'll talk about how to mitigate that in just a little bit.

So here's an example. Since EMs are set in a relative unit based on the size of the type, we have some CSS here that sets our paragraph at one EM beneath the font size of 100% set for the HTML root of the document. So therefore, that paragraph is 16 pixels. The H1 is set at three EM. Three times 16 is equivalent to 48. That's generally a pretty good ratio of heading to a body copy.



But we also want to remember, that if somebody needs to zoom the text, you still want to preserve the relationship so that's where the relative type size comes in. If you zoom in and you make the font size a little bit bigger for the root and you set it to some other percentage, then all the other math would go with it.

Here's an example. We have—on the left—a desktop screenshot. Paragraph set at 100%. And a paragraph set at 16 pixels. And then, a paragraph set of 16 points. Now the thing to remember is, a point measurement does correspond to a physical measure, but that's never been able to be represented accurately on the web. So it's really not appropriate for use.

The screenshot on the right is from an iPhone and you can see that while physically, it's slightly different than the 100% and 16 pixel on the left, it is still a readable line of type. And you still see that odd difference between pixels and points, even though they tend to be something that is thought of as interchangeable.

So let's take a look again, at what we have here on the screen. We've got—on the left—everything set in pixels with the width of 90%. And what we want to do is convert that pixels over to EMs. Now since we haven't done anything to change the route size of the type—which is still set it 100%—we can simply take these pixel values and convert them.

And to do that, we're actually using that same formula that we learned before—target divided by context equals the result that we want in EMs. So we have eight pixels—8 divided by 16 is 0.5 so our padding on the right becomes 0.5 EM. And the max width, instead of being 864—864 divided by 16 is 54 so it's 54 Ems for our max width.

```
.section-content {
   margin: 0 auto;
   padding: 0 0.5em; /* 8 / 16 = 0.5 */
   max-width: 54em; /* 864 / 16 = 54 */
   width: 90%;
}

.main-nav {
   max-width: 54em; /* 864 / 16 = 54 */
   width: 90%;
}
```

And we do the same thing with the main nav.

Now what we want to do is take a look through our CSS—scrolling up to the top—we actually want to scour all the way through and find all the places where we're referencing pixels and convert them over to EMs. And remember, the only place that you want to leave that alone would be in a border thickness, text shadow, or a box shadow.

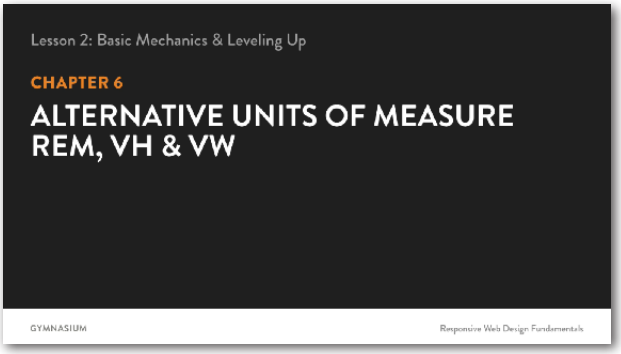
So I'll get you started. We'll take a look at the Layout section. Padding of eight pixels. You can change that to 0.5 EM. And down here, this 864, we just looked at that math. We know this is 54 EM. Now the 900 pixels—we'll go over to our calculator—900 divided by 16, 56.25. 56.25 EM, that gives us a nice relative size of type.

There's a few more places you'll need to do it. You're going to have to take a look up and down for font sizes, padding, border radius, and you can convert all of those things over to EM measurements.

So have a look. Go through the document. If you need to take a look at what the end result is, you can cheat a little bit and take a look at styles final.

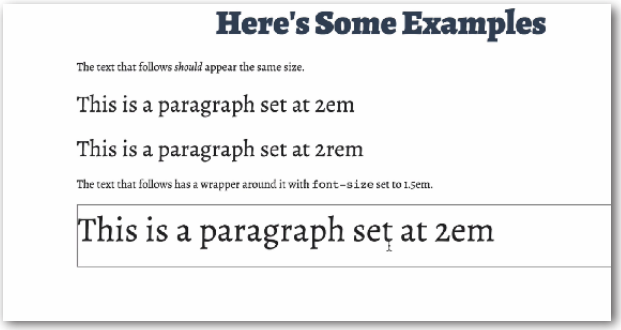
CHAPTER 6: ALTERNATIVE UNITS OF MEASURE: REM, VH, & VM

Let's talk about EMs and REMs—the newer, older kid on the block. First, an explanation, REM versus EM. An EM is a relative unit of type. And that means that one EM corresponds to whatever the size is that this type has been set. And anything that is set next to it would be a multiplier. So something that is 2 EM is twice the size of something that does not have the size added to it.



REM, or root EM, is something that's going to be set based on whatever is in the root of the document. So I'm going to flip over to the browser, and we're going to take a look at an example. So these two lines of text are each set, and they're set at 2 EM, or twice the size of the paragraph text that appears above. One is set in EMs. One is set in REMs. They're both coming back to that root font size of 100%, each one now being set at twice the size.

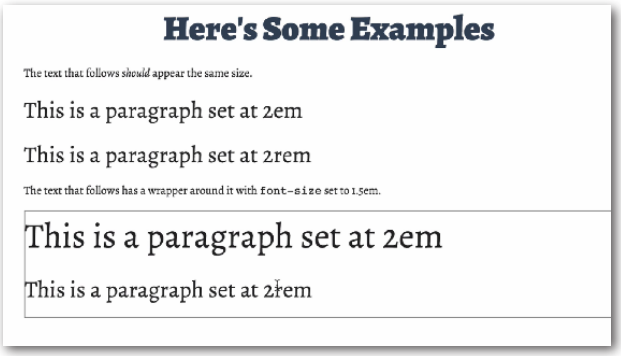
What happens with EMs is they're always set in the context of the immediate parent. So what we're going to do is look at the code, uncomment this next little block so we can play around with this a little bit. And we're going to cut this out of here for the moment and look at this again and reload.



So we now have a wrapper. And you can see that this paragraph is set at 2 EM, but it's physically much larger than the one above. And we need to look at why. We go back into the code, and we see that the div wrapping around it has a class of font step 2. So let's look at the CSS. And we'll see font step 2 has a font size of 1.5 EM.

And so if we look again at that HTML page, we can map this mentally. We have a paragraph at 2 EM. Its immediate parent is at 1.5. So we know that this is acting as a multiplier. So the div sets everything at 1.5 times the paragraph text above, and this is 2 times that immediate parent.

So it's roughly three times the size of the text above it, which is a little bit confusing, which is why a lot of people shy away from using EMs. There's a really simple answer to avoiding that kind of problem, which is don't set font sizes on the container. But we can come back to that later.



I'm going to undo. And we're going to leave this text set in here with the REM class. So size 2 REM, so root EM here, is always going to reference back to the root element. And so if we save this and go back to the browser and reload, we'll see that what's set in REMs is still the same as the one above, which is twice the size of the text in the paragraph. So we can see here the wrapper affects EMs. It doesn't affect REMs.

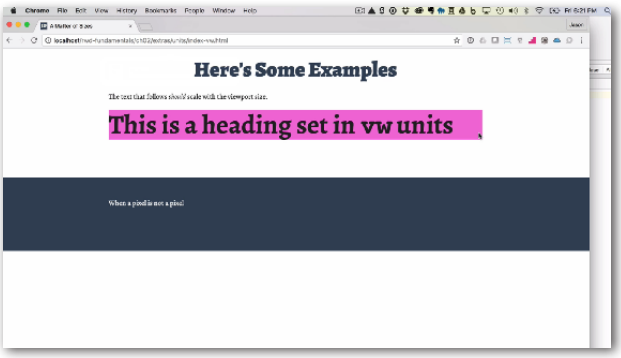
Now, the problem with REMs—if you have to support older versions of IE, they don't support REM, so you have to supply either a polyfill or some kind of fallback, usually set n pixels. I prefer to just, in my mind, use EMs properly. Only set a font size on the text element itself and not on the wrapper, and you'll never have problems with inheritance.

Now we're going to talk about the new new kids on the block—VW and VH, or Viewport Width and Viewport Height units. Now, viewport units are relatively new, but they are well-supported across all shaping browsers at this point. And that gives you a unit that is 1/100 of the height or width. You might want to mistake this for a percentage, but this actually works for height.

So what is interesting about basing type on viewport width is that it will scale with the width of the window. Really interesting possibilities for a known length string of text. Sometimes not as useful as you'd like when you have dynamic content. So let's take a look at what that means.

So here we have the code, and this is in the Extras folder in Units. And this is index-vw.html. And you'll see that we have some sample text and we have a heading that's set in VW units. And if we look at our CSS, we'll take a look, and it's really simple—font size 4.5 VW.

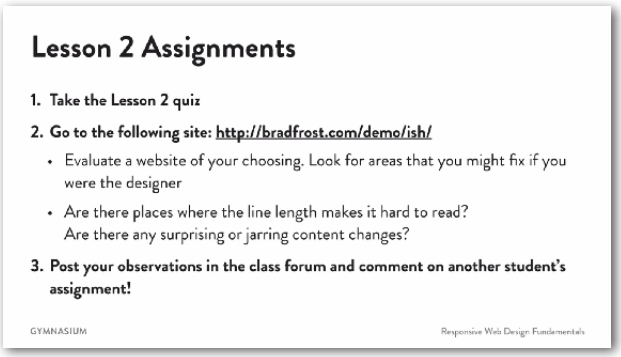
So it's going to take the width of the window, and it's going to use that unit as a multiplier on the type size. And we look at that in the browser. And it looks kind of like this.



I made the background of the div pink just so you can see what's going on as we scale. Now, bear in mind that the text is scaling based on the width of the window, not necessarily the div. So I'm going to go a little bigger, and we're going to go a little smaller. And it's always going to be proportionally the same amount based on the width of the window itself.

So this can do some interesting things. Where I find it troublesome is if you're using a content management system and you have text that's filling the entire width of the column isn't ever going to exactly work. I think it's really interesting for a layout. I think there's a lot of possibilities here. And we'll probably all be using it quite a bit more in the future when it's a little bit more fully supported. But I think for type, you really need to use some caution.

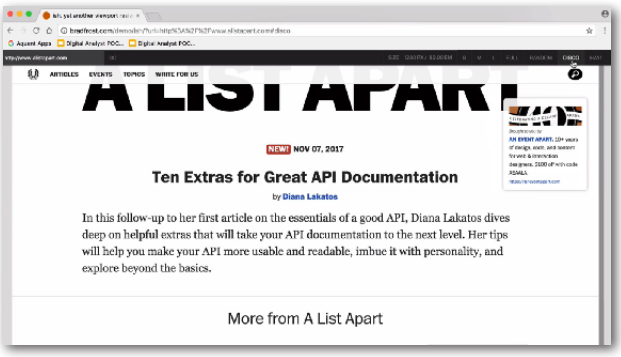
So we've finished going through. We've set EMs wherever we need them. We've gotten rid of pixels where they shouldn't be. We've played around with a few different kinds of measurement units. And we have a nice, responsive page.



Assignment time. As always, the first assignment is to take the quiz on the classroom site. The second assignment is to help you keep thinking about media queries and break points. For this assignment, I want you to go to the following URL.

This is the demo page for a viewport resizing tool developed by Brad Frost called Ish. You can read the details of what the tool does on the site. But here it is in a nutshell.

If you type the URL on the upper left-hand corner, you'll be sent to that site as expected. So let's try alistapart.com. You'll see there's a small toolbar at the top of the page. Also, if you place your cursor on the far right-hand side of the page, you'll see that your cursor turns into a double arrow. And if you click and drag to the left, you can see the page begin to resize responsively.



You'll also see the various breakpoints on the page where the layout changes. At the top of the screen, you have the ability to type in a specific pixel or m width, choose a small, medium, and large screen width, and a few other options, including the best one of all—disco, which will automatically keep resizing the screen back and forth until it's time to go home.

Your assignment is to use this tool to explore a website of your own choosing. If you're working on a portfolio of your own, this is a great way to start examining the breakpoints

and hopefully give you some guidance as to where things need fixing. Alternatively, you can choose another website and take notes on areas where you might choose to fix it if you were the designer.

Questions to ask yourself—are there places where the line length makes it hard to read? In the layout, are there any surprising or jarring content changes? Take note of all your responses and then go to the classroom site and make a post for lesson 2 with your observations. Be sure to include the URL of the site you visited. Additionally, find another student's assignment and give them feedback as well.

That's it for now. In the next lesson, you'll be turning your attention towards responsive typography and a dive into CSS columns in Flexbox.