

GYMNASIUM

---

# RESPONSIVE WEB DESIGN

---

*Lesson 10 Handout*

*Optimization & Performance*

# ABOUT THIS HANDOUT

This handout includes the following:

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

## CORE CONCEPTS

1. The word “testing” can mean very different things to different people so we break down responsive testing into three components: viewport (or breakpoint) testing, device testing, and cross-browser compatibility testing.
2. Viewport or breakpoint testing is primarily focused on detecting design or layout issues that show up on different screen widths. Testing is accomplished by slowly expanding or contracting the browser window or using a web application such as [responsivepx.com](http://responsivepx.com) and searching for issues. If a design issue is detected, the next step is to identify the CSS media queries and specifically the styles within a media query that are responsible for the layout problem.
3. Testing on devices is necessary because it identifies issues that basic browser testing will not uncover. Devices have additional capabilities that software can emulate but not reproduce, such as orientation, pixel ratio, monochrome, and more. Additionally, when using a device such as a smartphone you can also test bandwidth or network speed and its effect on your design. To accomplish this, many designers and developers collect popular mobile devices as a device lab and use this to test their sites.
4. Cross-browser compatibility testing is more complex with Responsive Design than with the “standard” web design. Much of this complexity comes from the fact that common elements in Responsive Design such as HTML5 elements, media queries, and related JavaScript are not well supported in the (still popular) older versions of Internet Explorer. Clear decisions about which versions are being supported as well as extra time assigned to testing are important.
5. The file size of the average web page is increasing daily, and one of the side effects is that some critics have stated that responsive websites are not practical from a performance standpoint, especially with devices that rely on slower network connections. To improve performance there are three steps: a.) Treat performance as another aspect of design, b.) Analyze file size of your page with a tool such as YSlow, and c.) Define a “performance budget” for your pages. A performance budget is a fixed file size for a given page that you choose not to exceed.

# ASSIGNMENTS

1. Quiz
2. Read “Performance As Design” article by Brad Frost  
<http://bradfrostweb.com/blog/post/performance-as-design/>
3. Test & Optimize Your Own Portfolio Page  
Check your site in older browsers as needed. Run YSlow on your site ([yslow.org](http://yslow.org)) and based on the results optimize as needed, most particularly your images as needed.

## RESOURCES

### READING

- <http://bradfrostweb.com/blog/post/performance-as-design/>  
Performance As Design
- <http://24ways.org/2012/responsive-responsive-design/>  
Tim Kadlec’s article on treating performance as a component of the user experience

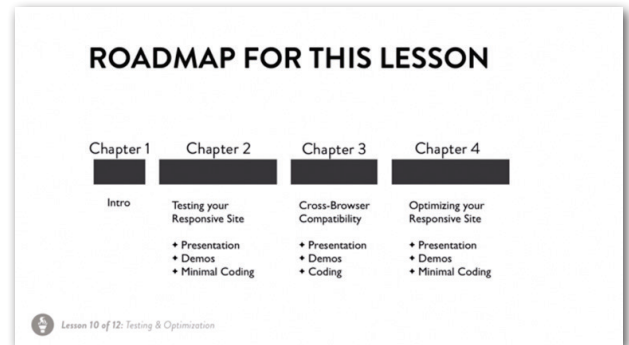
### WEBSITES MENTIONED IN THE PRESENTATION

- <http://bradfrost.github.io/this-is-responsive/resources.html#testing>
- <http://responsivepx.com/>
- <http://www.quirktools.com/screenfly>
- <http://dmolsen.com/2012/06/26/how-to-build-a-device-lab-part-1/>
- <http://html.adobe.com/edge/reflow/>
- <http://www.ie6countdown.com>
- <http://www.browserstack.com>
- <https://speakerdeck.com/paulrobertlloyd/the-edge-of-the-web>
- <http://httparchive.org/trends.php>
- <http://validator.w3.org/nu/>
- <http://jigsaw.w3.org/css-validator/>
- <http://www.yslow.org>
- <http://zoompf.com/2012/05/html5shiv-and-serving-content-from-code-repositories>
- <http://www.w3.org/TR/netinfo-api/>
- <http://24ways.org/2011/conditional-loading-for-responsive-designs/>
- [http://filamentgroup.com/lab/ajax\\_includes\\_modular\\_content/](http://filamentgroup.com/lab/ajax_includes_modular_content/)

# INTRODUCTION

*(Note: This is an edited transcript of the Responsive Web Design 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.)*

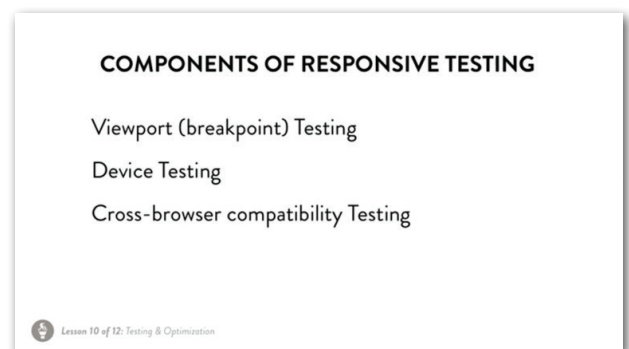
Welcome to Responsive Web Design, a free online course developed by Aquent. Responsive Web Design, or Promote Yourself Responsibly, Build A Portfolio For All Devices. This is lesson 10, Testing and Optimization. As always, at the end of this lesson, there will be an assignment and a brief quiz. At any given point, be sure to use the pause button in order to stop and work with code, , or simply to absorb a concept. And finally, if you have questions, be sure to hit the Forum. There, we have instructors TAs, and most importantly, your other classmates available to help you out.



Let's just talk a little bit about the road map for this lesson. Currently, we're in chapter one. And this is the intro. Chapter 2, Testing Your Responsive Site. There, we'll have the presentations and demos, and a little bit of coding for you. Chapter 3 is Cross-Browser Compatibility. Again, presentations and demos, and the bulk of the coding in this chapter. And then finally, chapter four, Optimizing Your Responsive Site. Again, a presentation, some demos, and a little bit of coding for you. This is Responsible Web Design.

## TESTING YOUR RESPONSIVE SITE

In this section, we talk about testing your responsive site. Just a little bit about the objective of the lesson: We're going to be presenting some strategies to help you test the design of your responsive site on multiple screens and devices. I do want to mention that I'm going to try to keep it focused on responsive testing and optimization. In other words, what sets responsive design apart from traditional design in the field of testing and optimization?



Now, if you remember at the end of the last lesson, we had a little bit of a party because we had finished the design of our layout. So we've been getting responsive layout and images and navigation. Everything is looking pretty cool, except now, it's the morning after. And things don't maybe look as good as we thought they did. After all, this is what we're heading to.

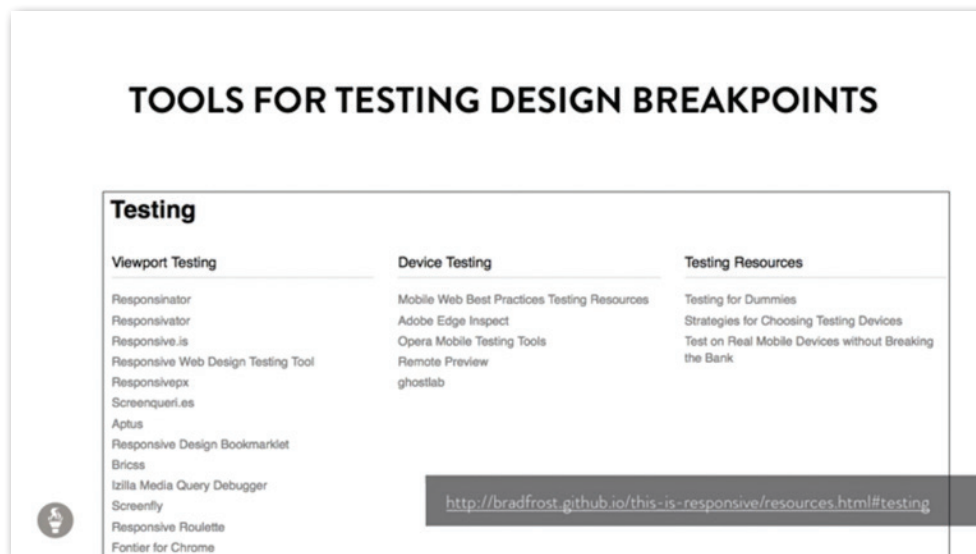
We're going to be launching our ship, or our site, shortly. And, of course, what we want to avoid is something like this. As it goes out to launch, we're asking questions like, it's going to work, right? Well, we want to avoid that shipwreck that we've seen before. And so that's what testing and optimization is about, making your site work as well as possible on all these different scenarios. Devices, screens, these are the things that we have to start worrying about, especially how our site looks on these different devices.

So what I want to do is break down some of the components of responsive testing. First, we'll talk a little bit about viewport or breakpoint testing, which you can think of as layout or design testing. Now, the next component, device testing, is related to the first. However, we're really focusing on hardware here versus just design and layout. And finally, cross-browser compatibility testing. How does your site look on different browsers?

So to make this a little more clear, let's take a look at the difference between viewport testing and device testing. So just to reiterate, a viewport is the section of a device or a screen that displays your website -- so displayed here in orange-- versus the entire device. And why does that matter? Well, because different devices have different capabilities. So for example, when you flip a phone on its side, it turns from portrait to landscape.

So let's explore this difference a little bit more and focus on troubleshooting breakpoints, or viewport testing. So obviously, we have different screens that we have to worry about. And the whole concept of responsive design is that your layout changes based on the size of the screen. So we want to examine this closely by looking at these breakpoints.

Now, I've mentioned this site before that you're seeing. This is Brad Frost's resources for responsive designs or responsive patterns. And what we're looking at here is the testing section of this site. And we can see here on the left that there's lots of viewport testing tools.



And we're going to take a look at one of these, responsivepx. I'm choosing this one because, well honestly, it's the one I've been using the longest. And as you'll see, it does a pretty good job of what we're trying to accomplish.

Why this tool and not the others? Well, first of all, it's just a matter of time. We don't really have that much time to walk through all the different tools. But the other part of it is they generally do the same thing. So I'm choosing this one because I'm comfortable with it. So let's go ahead and take a look at how this works.

This is a site that's online, [responsivepx.com](https://responsivepx.com). And you can go to the site right now if you choose or after this presentation. And now, the first thing that you need to do is make sure that your site is online somewhere. It has to be hosted so that you can take the URL and punch it into this field up top. Once you do that, your site is going to load in the middle of the screen here.

And there's some handy sliders that will allow you to enlarge the width of the screen and the height of the screen. And you can see your breakpoints take place in real time. This is exactly what we're looking for. We're trying to identify those points that don't work or there's something going wrong with them. And we can see one of those problems right here.

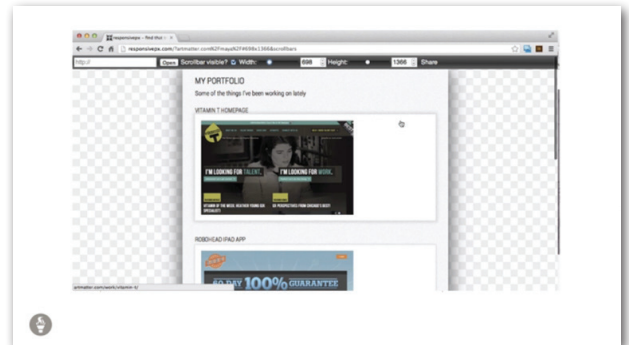
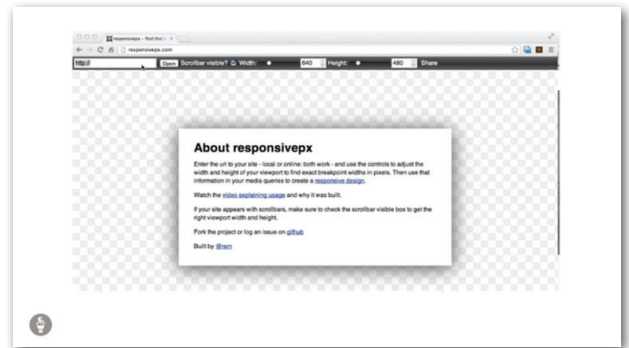
If we scroll down, we can see that this box that contains these three figures under the portfolio section is extending the entire width of the screen. But it doesn't happen all the time. In other words, if we begin to change the width of this slightly, we're going to see then right around here, 475, things still look good. But as we begin to increase the width, that's when that box begins to stretch. And we don't like that. It doesn't look good.

So we've identified a problem here. And specifically, the problem is that the images in that section are reaching their maximum width at 25em or around 400 pixels. However, that parent figure element, the one that contains the image, continues to be fluid. And that's the problem that we just saw. So time to fix it.

Let's go into our lesson file, `portfolio_start.html`. As always, we want to Save As and name this `portfolio_work.html`. And this creates a backup. And then, let's go ahead and identify the section that we're going to work with. We have to scroll down here to about line 84. And you can see the first figure element. And that contains this image of `vitamin.png`.

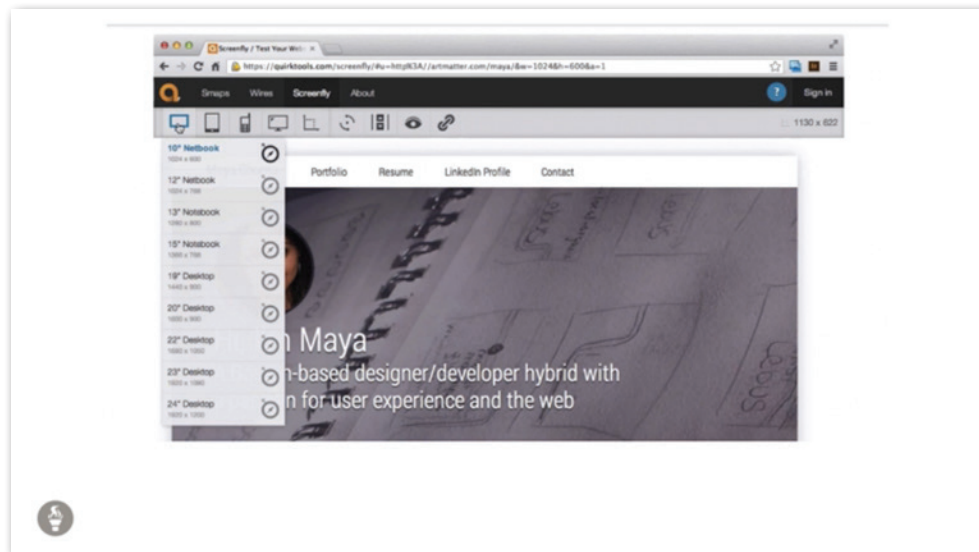
Let's go ahead and open up our external stylesheets. And I should point out that I took all of the styles from the last lesson and put them into this stylesheet for convenience sake. In fact, we're going to be working with just the external stylesheet from this point on.

Inside that stylesheet, you want to find the media query here, `@media screen and {max-width 45 em}`. This is for screens less than 720 pixels. What we're going to do is we're going to add a new style rule for portfolio figure. And the declaration here is going to be `{max-width: 25em}`. So that ensures that the figure element never gets larger than 25 em. And this is going to hopefully take care of that box problem.



Let's go ahead and save our document. Let's test it in the browser. And let's make sure that this is actually doing what it's supposed to. Now again, we don't actually have to put this back online if we don't want to. You can test this locally.

But at some point, you're going to want to test it again remotely. And we're in good shape. That box issue has been solved. And this is just an example of how you can begin to use some of these testing tools to identify layout problems and then solve them.



Now, as I mentioned before, there are dozens of these types of tools here. I promise not to go through all of them. I will show you one other. This is called the Screenfly. And the way this works is similar. You have to put the URL of your responsive site into the web application here. But then you can choose to display this on different types of devices.

So this is, obviously, emulation. However, what it does allow you to do is to see your site as it looks on a netbook or a Kindle Fire. And we can even go into the mobile section here. Let's see what this looks like in the iPhone 5.

And if we want to see what it looks like in the portrait view, we can rotate it. So that's kind of cool. And we can allow scrolling. So we can actually interact with this and make sure that everything is working the way it's supposed to.

Additionally, if you want to, you could even put in your own custom resolutions here. And again, this is just a nice little icing on the cake. So this is a great service as well. So while both these services are pretty cool and extremely useful, in the end they're limited in that you want to test on a device. You need to test on hardware if you're really committed to making sure that your site is going to look great.



So we really have to pay attention to the plethora of choices that we are currently faced with and will continue to be faced with in the future. And in fact, this is really one of the key aspects of responsive design in the first place. We're designing things for devices that don't even exist yet. So what are some of the things that we're looking for?

Well, one of the main differences between a device and desktop screen is we have this portrait versus landscape issue. So you don't turn your computer monitor on its side. And even if you did, it wouldn't recognize that it's on its side. This is what devices do.

We also have touch versus no touch. So even within the ecosystem of devices, we have ones that are touch-screen and ones that are not. And finally, there are other capabilities that we need to pay attention to. On the left, we've got an HD or Retina for iOS and OSX. And so this has a high resolution screen. And on the right, we've got something like one of the Kindle Paperwhite devices. And this is simply a monochrome device.

And we could actually target this with a media query if we wanted to. But at the very least, we need to know that this device exists and someone might be seeing our site in gray scale. And this is the one thing that everyone pays attention to, something we're going to cover a little bit more in depth in the third chapter on optimization.

But we're simply identifying, here, that connection speed is one of those capabilities that we have to think about. And the word design here is not just visual design. It's the entire experience of your site. So that's what we're worried about.

So one of the realities of a responsive developer and designer is to have a testing lab. I'm going to get into all the details of how to create a testing lab. So this gentleman, Dave Olsen here, has a extremely thorough explanation of how to build a device lab.

And I highly recommend you take a look at the details at this URL. And it's down there at the bottom of the screen, as well as a link in the classroom. All sorts of good advice here. I'm going to give you the executive summary.

These are the top six devices that Dave recommends most people have inside their labs. And number 1 is an iPod Touch. 2 and 3 are mid-level and high-end Androids. You want to have one, at least, that runs Android 4.0, because we have different operating systems to worry about there. And then you should have some form of a tablet, whether it be iPad, Android, or Kindle. And finally, there is BlackBerry and Windows Phone 7. And again, these are not as high a priority as number 1, 2, 3, and 4. But they're good things to have just to make sure that you understand what your site looks like on a diversity of devices.

## EXECUTIVE SUMMARY

1. iPod Touch
2. Mid-level Android
3. High-end Android: (capable of running Android 4.0)
4. iPad/Android/Kindle Tablet
5. High-end BlackBerry
6. Windows Phone 7

Lesson 10 of 12: Testing & Optimization

```
<!DOCTYPE html>
<html>
  <head>...</head>
  <body lang="en">
    <div id="container-nav">...</div>
    <div id="container-header">...</div>
    <div id="container-content">...</div>
    <div id="container-footer">...</div>
  </body>
</html>
```



Now, assuming that you have a device lab set up, testing is not always fun. So I'm going to show you at least one method here on how to minimize your testing time. Now, the example I'm going to talk about here is called Adobe Edge Inspect. And if you have Adobe's Creative Suite or Creative Cloud software, if you've already paid for this, then you have this program already and you can use it for free.

If you don't have it, then you can pay a little bit for it. And there's alternative applications out there as well that do something similar. But as you'll see in a second, this one is pretty slick. And let me show you how it works.

Essentially, we've got a laptop in the middle. And this laptop has already been synchronized wirelessly with an iPhone on the left and an iPad on the right. Now I could actually have added other devices. And this is one of the key aspects of this program. Essentially, what it does is it links up multiple devices. And then you can run and then test them from one central location, your desktop or laptop.

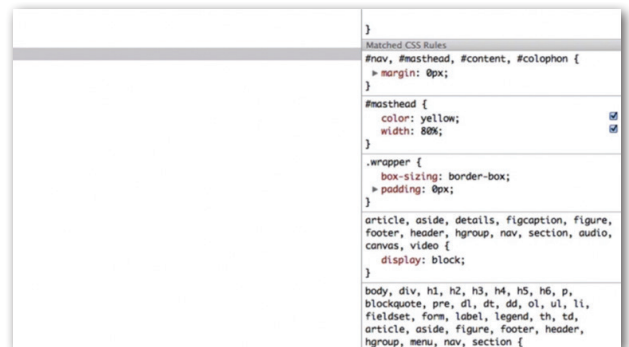
So let me show you a little bit more how this works. Again, you'll see here that everything's in sync. So as I began to click on links on my laptop, you're seeing the phone and the tablet respond in virtually real time. And, of course, they both have their different layouts. And you can examine them.

Now, there's a little bit more to this too. In addition to just being able to detect problems, you can also inspect them, hence the name. And so the way this works here is you can see these two devices are currently hooked up. I'm going to go ahead and click on these little brackets here to access the iPad. And then I'm going to dive in.

And now, it's switched over to the screen. And the way this works is very, very similar to browser developer tools like Firefox's Firebug, or their built-in developer tools, or Chrome's developer tools, or Safari's, or Microsoft's. All the browsers these days have something very similar to this and they all work essentially the same way.

You'll see here as I begin to hover over these different divs, I'm going to switch over to the video in a second. And you can see that these are lighting up in a real time, just like a browser's developer tools. This is very cool. And additionally, just like a browser's developer tools, I can go into the stylesheets and enable or disable them, or modify, like I'll do here. I'll change this from white to yellow.

And this is just enough to show you that it works. We're not going to be doing much more in here. And there we go. Now, it's turned yellow. So this is a great way to speed up the amount of time you're spending on these different devices. You can do them all at the same time. Well, you've just cut your time in half or more.



## CROSS-BROWSER COMPATIBILITY

So that's it. We're all done with testing. We must be all done, right? Wait, what's going on here?

*"They're coming for you, Barbara."*

*Stop it! You're acting like a child.*

*Look-- they're coming for you. Look-- there comes one of them now!* "

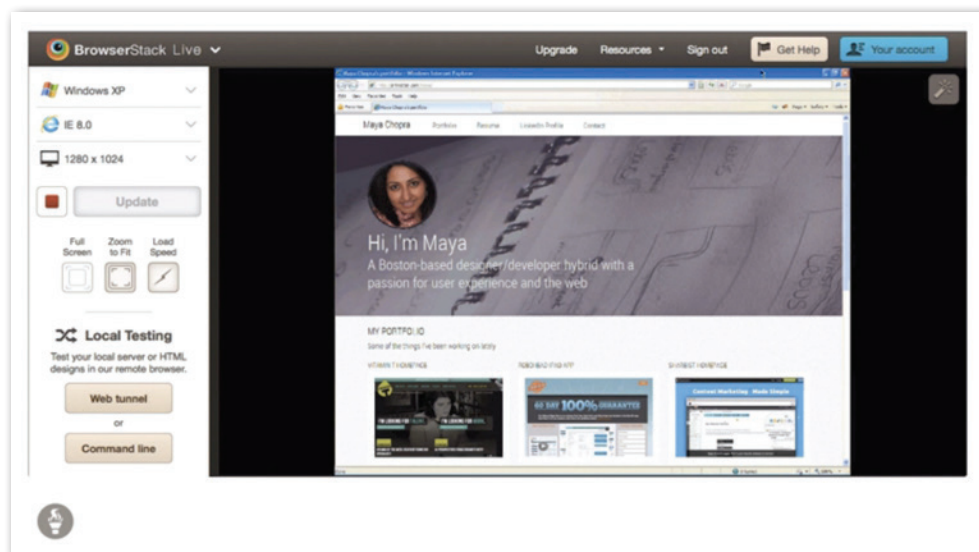


Ooh, yes, they are coming for you. You thought you were going to escape, but you're not. This is something that developers have had to work with for many years, and this is the specter of IE6, 7, and 8. And we have to talk a little bit or a lot about the implications of this, depending on your personal situation.

So let's just take a look at some of the facts here. This is the web browser market share as of spring 2013. And we can see here that Internet Explorer has the lion's share of the market share. These are all versions of IE, and it's about 44% currently.

Now what's equally important is to be able to dive into the specifics of this chunk here. So what versions of IE are inside this 44%? Well, let's go ahead and take a look at that. And what we can see here is that 50% of that 40% is IE9, 40% of that is IE8, then we've got IE7, which is around 7%. And this little tiny blue wedge that I can barely click on is Internet Explorer 6, that 0.6%, according to statowl.com, which is the site that I'm using this from.

So these are all really valuable numbers because this tells us how much time and effort we need to put into testing for these different versions of IE. Clearly, we have to pay attention to IE9 and most likely IE8. But then IE7 and 6-- well, that's going to be a judgment call, and we should talk a little bit more about that.



IE6, of course, is the one that we're used to dealing with if you've been a developer for a long time. Even Microsoft itself is trying to kill IE6. This is a website called [ie6countdown.com](http://ie6countdown.com), sponsored by Microsoft themselves, and they're currently showing you how fast IE6 is disappearing from the world.

And again, Microsoft has absolutely nothing to gain from having IE6 hanging around. They want you to use their new browsers, IE9 and 10. Both of those have extremely good support for things like media queries, CSS3, and other aspects of responsive design that we're using.

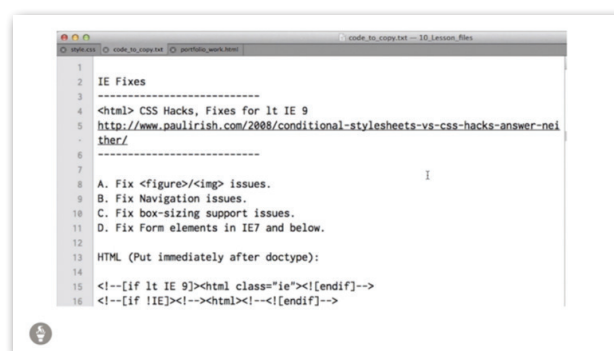
Now I don't want to get too far into this, but if you do you have this conversation-- and this is a conversation here between Alexander the Great and Diogenes the philosopher. But I'm going to imagine that this fellow here is the client and this guy in the barrel is the developer. And essentially when the question comes up of whether or not you should spend significant time on fixing your sites for things IE6 and 7 and 8-- well, this is one of the responses that I might say. Would you rather pay me three days to fix IE, or would you rather pay me three days to add some new and cool features for modern browsers and devices?

Possibly if you frame it this way, you might be able to avoid spending time fixing IE6 and 7. There are limited returns there, and you could have a much bigger impact, perhaps, by adding new features for devices that people are using today.

Having said that, we are creating a portfolio site. So here is our sample portfolio site. You're pretty familiar with it. And we have to think about the context of a portfolio site. Let's just take a quick look at what this might look like on a couple different versions of IE.

So I'm using a site here BrowserStack, and there are many other sites around here like this. Essentially what it's doing is in real time it's taking a look at your site through the eyes of different systems. So in this case, Windows XP, IE8, at 1280 by 1024. And the question is what does that site look like in that configuration? As we scroll up and down, I'm actually pretty happy. If this was my target audience, then I would be OK with this. And we might want to double check this and say well, what does this look like at 800 by 600? And what BrowserStack will do is refresh the page and show you this view.

And as we just saw, IE8 still is a pretty big market share. But let's go ahead and let's test it on IE6. Let's update it and let's see how it plays. And the answer here is well, it's not going to be quite as good. We can see that this whole Checkbox Hack that we added in a previous lesson is really falling apart. The checkbox here is visible, and if we scroll down we're probably going to find some other issues. The layout is a little wacky, and it's usable but really not cool.

A screenshot of a code editor window titled "code\_My copy.txt" showing HTML code for IE fixes. The code includes a list of fixes (A, B, C, D) and conditional HTML code for IE 9 and IE 10. The code is as follows:

```
1 IE Fixes
2 -----
3 <html> CSS Hacks, Fixes for IE 9
4 http://www.paulirish.com/2008/conditional-stylesheets-vs-css-hacks-answer-nei
5 ther/
6 -----
7
8 A. Fix <figure>/<img> issues.
9 B. Fix Navigation issues.
10 C. Fix box-sizing support issues.
11 D. Fix Form elements in IE7 and below.
12
13 HTML (Put immediately after doctype):
14
15 <!--[[if lt IE 9]]><html class="ie"><![endif]]-->
16 <!--[[if !IE]]><!--><html><!--<![endif]]-->
```

So the question here is how much time do you want to invest in making this look good in IE6? And we do have to think about the context. So again, we've got a portfolio site, we're advertising ourselves as a designer and a developer. Maybe it would make good sense to show a prospective employer-- so, someone like this guy here, who might be our boss, we might want to show him yeah, we can actually make a site that looks good in IE6, if we have to.

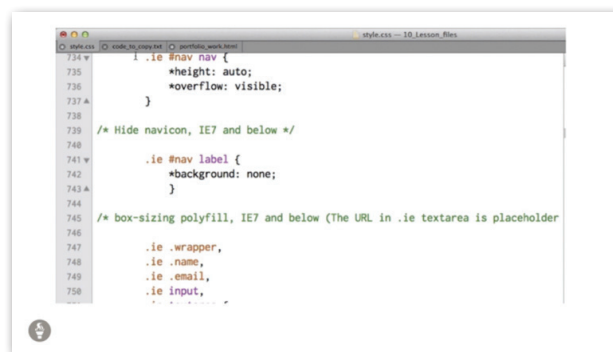
Now again, you can make that call on your own for future projects. But what we are going to do is add some IE8, 7, and 6 fixes very quickly here in this next section of coding.

So you'll need to open up your document portfolio work.html. And one thing I do want to point out is we have removed the internal styles here on purpose, and the reason for that is everything at this point on should really go into our external stylesheet. Additionally, we've got a document code\_to\_copy.txt, and this is something that you're familiar with if you've worked on previous lessons.

What you want to do is open up this document, and we're going to be doing an awful lot of stealing here. We're going to be taking all this code that I have created for you and we're going to be pasting it into our external stylesheet. And this is going to fix a number of issues in IE8, 7, and 6.

Now what I'm not going to do is go through this line by line, and I'm not going to spend a whole lot of time on the details of each one. I will point out this URL here at paulirish.com. This is the foundation of a lot of what I'm doing here.

The first thing that we're going to be doing is taking this HTML conditional comment here. So, conditional comment is code that is targeted specifically for different versions of IE. In this case, we are targeting versions of IE less than IE9, and we're going to be adding this `<HTML class="ie">`. To make a long story short, any class that has ".ie" or the class "ie" appended to it will be targeted specifically for those browsers.

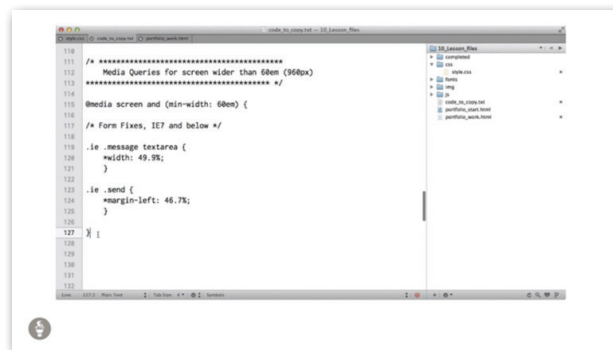


So having said that, we need to put this code in the top of our document in order to make this work, so copy those entire two lines. Go back to your portfolio work.html and paste it directly after the doc type so that it's at the very top of the document.

Now we're going to take this whole mess of CSS code-- and as I mentioned before, we're not going to be spending too much time on it. The only thing I do want to point out is this line here where it says absolute-path.com/js/boxsizing.htc. We actually mentioned this in a previous lesson. This is just a dummy URL. It is extremely important if you want this to work and if you want to support these browsers that you replace that dummy URL with a real one so the boxsizing.htc file, which is in your lesson folder, goes onto your site.

So after we copy all that code, let's go ahead and paste at the very bottom of our external stylesheet. I'm just going to clean this up a little bit and indent it just to make it obvious.

So I mentioned I'm not going to be talking too much about the details. We did try to put in some comments here so that it gives you a little bit of context if you're trying to understand what's happening. But for the most part, we're going to leave it alone.

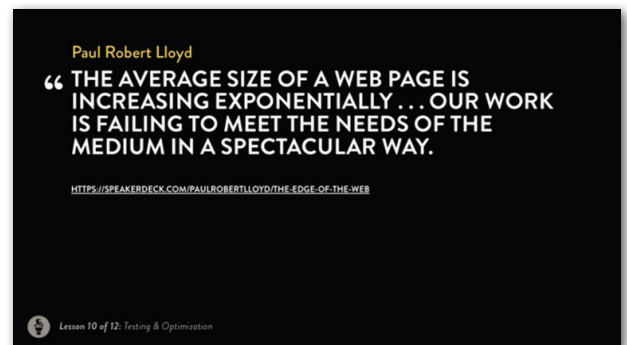


Now we've got some other stuff to do. Most importantly, we need to support the Advanced Checkbox Hack that's being used for the navigation of our site. So let's go ahead and take these two conditional comments, one for IE9 and earlier. This is pointing to jQuery, and another one which is pointing to an external JavaScript called checkbox.js, which is actually inside your JavaScript folder. You can see that here.

OK. Now we're going even further here and we're going to taking this attribute selector support. So again, this is related to the check box navigation. So we're going to take this conditional comment for IE6 and we're going to paste it there as well. And this is pointing to a JavaScript library called selectivizr, which again, if you want to do some more research on that, there's a link to what selectivizr does in the classroom links.

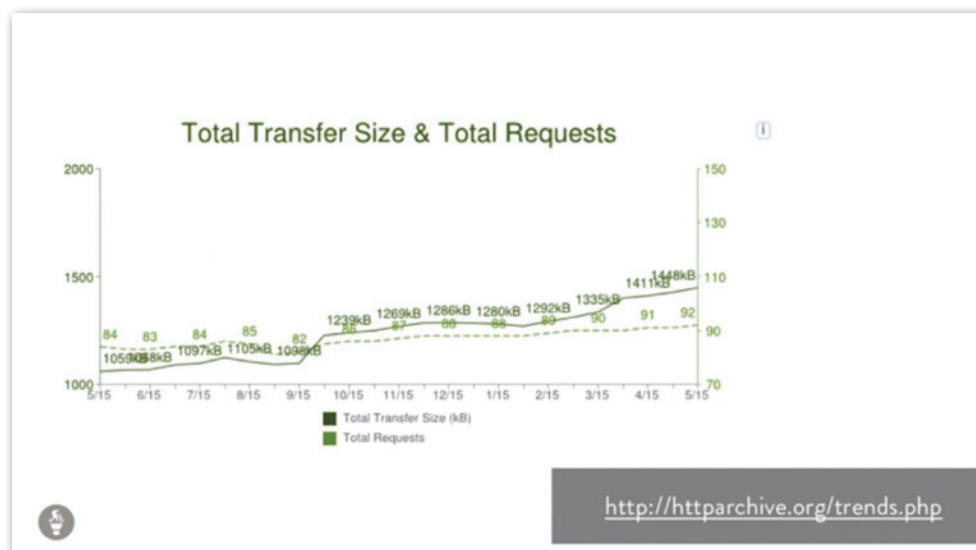
And finally, one of the other things that we noticed in that preview of IE was that the form was kind of a mess. So what we're going to do here is we're going to take these fixes for the media queries and we're going to paste them into our external stylesheet. So this is going to fix IE7 and 6 and make that form look a little better on those browsers. The only catch here is, again, we're doing this for our media queries, so be sure to find this media query minwidth 60m, and this is for screen sizes wider than 60 m or 960 pixels.

Coming up next, we'll talk about how to optimize the performance of your responsive sites. This is Responsive Web Design.



## OPTIMIZING YOUR RESPONSIVE SITE

In this section, we talk about how to optimize your responsive website. Let's go ahead and start with a quote from Paul Robert Lloyd, a developer for a firm called Clearleft. Mr. Lloyd says, “The average size of the web page is increasing exponentially. Our work is failing to meet the needs of the medium in a spectacular way.”





So, a great quote, and from a great presentation. There's a link down below. Highly recommend you download it, read it. It's great stuff. More importantly, it speaks exactly to the point of this lesson. What we're trying to talk about is the importance of testing and optimization. So what exactly is Paul talking about here? Well, he goes on to reference this graph, which is from [httparchive.org](http://httparchive.org). It's not a very pretty graph here, but the information is really important.

And what it's saying here is the average web page size is about 1448 kb, or 1.5 megabytes. And not only that, but this number has increased dramatically over the past 12 months, by almost 500 kb, so this tells us that web pages are getting bigger and bigger virtually every day on an average.

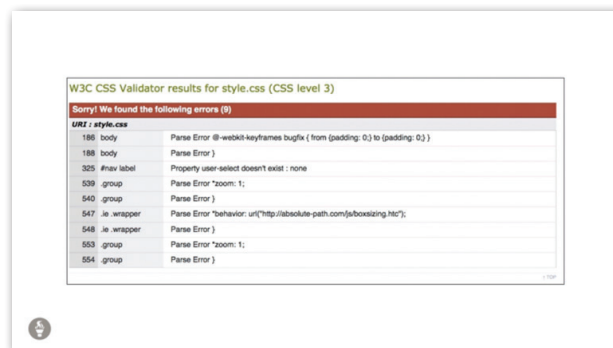
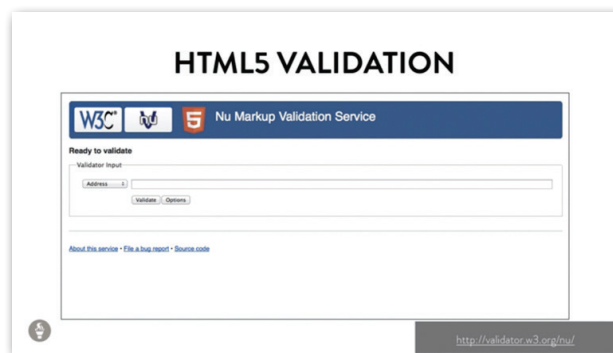
And this is not a good thing. And in fact, what's happening is responsive design is beginning to be painted with the brush a little bit of poor performance. So there are critics out there of responsive design who say that the sites are not particularly well-suited for the mobile environment, and maybe we should be using mobile-only sites instead.

Well, I'm not going to enter the fray of that argument right now. However, the point of this lesson is that through testing and optimization, we can dramatically reduce the size of our web pages. And that is a worthy goal no matter what you're trying to do. So having said that, let's talk about the concept of perceived performance. So this is the amount of time a user thinks it takes to load a page, based on when they click on a link or enter a URL into the browser, and when the page becomes usable.

And so there's a number of factors that affect that. Now one thing I should mention before we start getting into the details, here is our proverbial can of worms. Although they are gummy, because I didn't want to get too disgusting. But the point here is performance is a huge topic. And we could easily have a six hour course on that. This is only a few minutes. And I'm going to try to give you some of the highlights, and specifically, some of the details relating to responsive performance.

So before we do that, a little preboarding talk, because you validate your HTML and CSS right? So this is something that you should be doing anyway. And it's almost a necessary step before we even start talking about optimization. So let's just go through a few steps to make sure we're on the same page.

There's a service out there at [validator.w3.org/nu](http://validator.w3.org/nu). And this is specifically for HTML5 code, which is what we've used on the site. And the idea here is you can upload your HTML source code, and you get back some results as to whether it's valid or not. And what we want to see is green. We want to see this. So if we can eliminate the fact that we've got syntax errors or other issues, then we can continue on our way for optimization as needed.



It's the same thing with CSS in that if we validate our syntax, we might get some results that cause us to look twice. And this is the case here. So if you validate your portfolio page, we were getting nine errors. We don't necessarily need to get 100% for CSS. And let's just talk about why.

So in a previous lesson, we talked about this fix for iOS and Android. So what we're seeing here is an error getting spit back that says, well, we've got this WebKit keyframes bug fix code. And that doesn't really make sense. Well, we're OK with that, because we know we had to use it for our navigation.

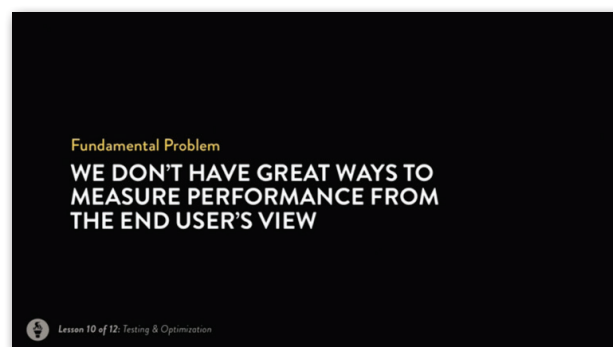
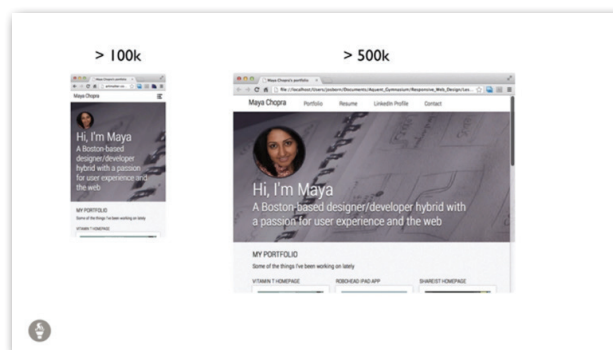
Same thing here. These errors with zoom one are things that you may have seen if you tried to validate CSS before. These are errors relating to IE hasLayout fix. And again, I'm not going to get into the details of this. We talked a little bit about it in lesson 4. And lastly, there's this box-sizing fix from lesson 6. And there's an error here as well.

The point with all this is we don't have to solve these. We just need to make sure we understand why they are there. OK, now that we've covered that, let the games begin. It's time to attack. So let's talk a little bit about how you attack the performance of your page. The first step is to treat performance as another aspect of design. We'll talk about that in a second.

Second step. Get the data with a tool such as YSlow. And we'll be talking about this in a second. And the third step is to set a performance budget. So let's go ahead and break these down. So performance equaling design. The point here is that when we talk about graphic design, we often have all these components, such as shape, line, color, and so forth. And these are accepted components of graphic or web design. This something that we all take for granted. You go to school for it.

However, what we're saying here is that performance is as much a component as these other elements. We might not think of it, but that's the way it is. Now I wish I could take credit for this concept. However, our friend Brad Frost at [bradfrostweb.com](http://bradfrostweb.com) has an excellent article called "Performance as Design." And he goes through the concepts of this quite well. I'm going to ask you to read this article at the end of the lesson.

But moving on, testing your site's performance. We're going to go ahead and do this in just a moment. There's a number of different tools. We use YSlow. And I'll talk about why we do that in just a moment. And then the last one is setting a performance budget. So a performance budget. What exactly is it? Well, it's very similar to a household budget. And the way it works is you essentially give a number to any given state of your page.





So you might say, well, my desktop view should have no more than 500 kb download time or page size. And my mobile site should have no more than 100 kb. If you stick to that budget, you can begin to get creative in the ways that you optimize your site, or maybe reduce the size of images, or any number of other aspects.

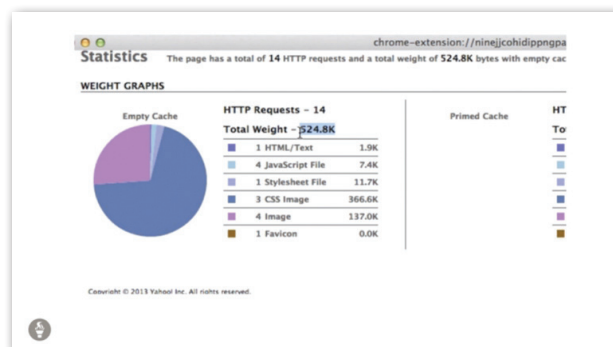
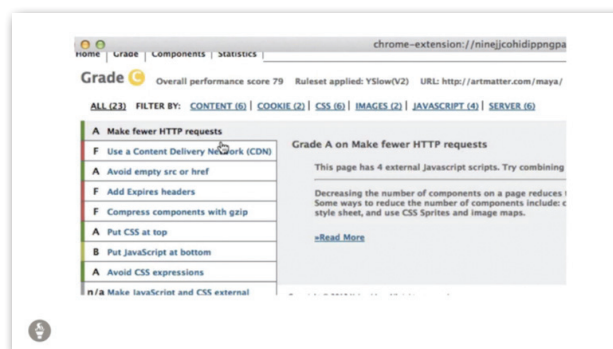
So with those three components in place, let's talk about one of the fundamental problems of performance. We don't really have a great way to measure it, or at least from the end user's view. That's because there are so many variables to consider, all we can really do is measure it from our perspective and then apply it to other perspectives. What we're going to do is we're going to use this technique.

So there's a website, [yslow.org](http://yslow.org). And this is a site that stemmed from the folks at Yahoo! who have had to deal with optimizing sites for quite some time. And the way that this works is you install a small bookmarklet in your browser. So go to [yslow.org](http://yslow.org), and you can do that very quickly. Once you do that, you'll have a small bookmark it the very top of your browser, such here in Google Chrome. And to run it, you simply click on that icon. And we get this little panel to open up. And then we can begin to test.

And what's going to happen here is a number of tests are taking place, and we get back a grade. And our grade here is 79. I say C. I think that's a C+ really. So, for example, we can see we got an A for making fewer HTTP requests. But we've got an F for using a content delivery network, or in this case not using it.

So we're going to explore this in just a bit. But what I want to cover before we go too deep into that is another useful component here. And that is the statistics. So the statistics are very useful. What we can see here is the total weight of the page. Instead of the average 1,500 K, we're about 500 K. It's not super great, but it's much, much better than the average. And so I'm happy with that.

Now if we're curious about the different components, like so where exactly are these numbers coming from, well, we can actually dig in a little deeper. So we get an overview here for the CSS image and the HTML image, but we can dive down even deeper by going into the Components section. And here we can break it down by toggling this part open. And it actually lists the individual images and the size. And we can click on them. And we can see that's one of the main culprits.



Size	Type	Component
1.9K	HTML	1 HTML/Text
7.4K	JavaScript	4 JavaScript File
11.7K	Stylesheet	1 Stylesheet File
366.6K	CSS Image	3 CSS Image
137.0K	Image	4 Image
0.0K	Favicon	1 Favicon

So this background image of the notebook is fairly large. And again, we talked quite a bit about that in the Responsive Images lesson. So this is essentially a high resolution image that will work well on things like the iPad. But it comes at a cost. So the other thing we want to test, though, is we actually want to get away from the desktop concept, or the desktop model. And let's bring this down to a single-column mobile view.

And let's test this again. So it's going to run it again. And what we're going to see is we get the same score. But the statistics are the ones we're interested in. Hey, look at that. We got 148kb for the total weight. So the desktop version was, again, around 500kb. The mobile version is 148kb. How did that happen? Well, we actually discussed this a little bit in Responsive Images.

And this is exactly the goal of that entire lesson. What we tried to do there was minimize the size of our images without compromising the design. And so here, we can break this down again, just like we did before. And this image is a much smaller one. So if you remember back to that lesson, we swapped that out using a concept called picture fill, which uses conditional loading, another concept we'll be talking about in just a moment.

OK. So let's go back to the grades here. And what I'd like to do is analyze why we got these F's. So we got a number of A's and B's, and that's cool. But why is our grade only a C or C+? So what I'm going to do here is walk through those four or five F's, and just talk about them briefly.

So the first one we got was using a content delivery network, or a CDN. What is a CDN? Well, essentially it leverages the fact that the user's distance from a web server impacts response times. So what does that mean? It means if you host your images, or video, or scripts, on a separate server, a user closer to that server will get that content faster.

So many organizations will actually pay for a CDN to serve all of their images. And in this way, a user will be able to have the perception that the site is loading faster. Because in fact, it is. They're getting that content quicker, because that content is spread over multiple servers throughout the world, perhaps.

We're going to talk a little bit more about CDNs in a second. But let's look at these other ones quickly. The expires-header error.

So the first time a visitor comes to your page, they download all of the files on the pages that they visit. And what we want to avoid is the next time they come to the site, we don't want them to have to do that again. So if we use the special code here called the Expires Header, we can actually make these files cacheable.

In other words, they're stored on the user's local system for a certain amount of time. In this example, we can see we're saying, well, the JavaScript should stay cached as soon as it's accessed plus a whole year. So the next time they come to the site, they don't need to download it. Everything goes much quicker. The next to fail we saw was compressing components with Gzip. So compression essentially reduces the amount of time that pages need to load, because they're much smaller.

### THE ARGUMENT FOR A CDN

```
<!--[if lt IE 9]>
<script
src="//ajax.googleapis.com/ajax/libs/jquery/1.10.0/jquery.min.js"></script>
<![endif]-->
```

- ♦ There's a high probability that someone visiting your site has already visited a site using the Google CDN.
- ♦ Therefore, the file has already been cached by your browser and won't need to be downloaded again

Lesson 10 of 12: Testing & Optimization

### THE ARGUMENT AGAINST A CDN

#### html5shiv and Serving Content From Code Repositories

Posted: May 11, 2012 at 4:20 pm

There are a lot of interesting findings that come out of my analysis of how the latest Top 1000 in using HTML5 compression. One finding was that JavaScript is the most common type of content served without compression. I hypothesize that this is due to websites linking to all these 3rd party JavaScript libraries and widgets. Normally a developer links to a 3rd party file to enable advertising, web analytics, user feedback and chats systems, or even social sharing widgets. But you cannot control these systems, and so those resources might not have compression enabled. So was the file on your own website are enabled for compression.

Today I want to focus on a 3rd party library which wasn't using compression because it leads to an important and often unmentioned performance lesson. Don't link to resources inside of source code repositories.

html5shiv is a JavaScript library that enables Internet Explorer 8 and earlier to understand and properly render new HTML5 tags like `<nav>` or `<aside>`. This is incredibly helpful because it allows websites to use these new semantic elements while retaining backwards compatibility. While html5shiv is open source and free to be copied and used anywhere, websites often link to a specific URL: <http://dmitrybaranovskiy.github.io/html5shiv/>

In fact, the html5shiv library is hosted on the library front-end project, which references this URL: <http://jsmamp.com/2012/05/html5shiv-and-serving-content-from-code-repositories>

Lesson 10 of 12: Test

Gzip is a specific type of compression. If you do it right it can save up to 50% or 60% of the file size depending on which technique you use, and the content of the file. And then we have this fun element, configure entity tags, or ETags. This is related to caching. So entity tags are method used by servers in a browser to figure out whether something has been cached or not. Or more specifically, whether something in your browser matches the one that's on the original server. If you have your ETag set up right, then this means that you're not going to force your browser to download new stuff. Therefore, everything moves quicker.

Now if we look at those four issues, which of those have to be done on the server, and which can be done on the client? And the answer is, all of them have to be done on the server-side. Well, with the exception of the content delivery network, kind of, sort of, we're going to take a look at that in a second.

So what this means is within the context of this course, really, these four elements are out of scope. We can't really do much on the server-side here. We're more interested on the client-side. So what would be great is if we could run that YSlow test again without those four server-side developments. And that's exactly YSlow allows you to do.

So here, I've checked of those four items. And I'm going to run it again. And I got an A. Woohoo. We got a 98. This means we've been doing things right. And that's one of the goals. So this is pretty cool. Now, if we're being picky, though, what about that client-side fix? What about that content delivery network stuff?

Let's just quickly go over the pros and cons of this. So we actually have some code like this in our portfolio site. So we've got a conditional comment that references jQuery. And this is sitting right now on Google's servers somewhere out in the world. In fact, it's actually probably sitting on multiple Google servers.

And the idea is this. There is a very high probability that when someone comes to your site, they've already been to a site that has this exact same jQuery code. If that's the case, that file has already been cached, and therefore, it doesn't need to be downloaded again. And that's good. So that's a cool thing.

Now there's arguments against using this. I'm not going to get into detail here because it gets a little thick and crazy. But essentially, this URL will go into the details. And to summarize it, they're basically saying that CDNs are not really doing it right. And so that caching and the compression is not handled properly, and that you're better off hosting it on your own.

And there's also another argument to be made when it comes to CDNs and responsive images. So they don't really play well together, like our two friends here. So a CDN's job is to serve images. But they don't necessarily serve responsive images particularly well. They're designed for static images. And therefore, there are some folks who say, you know, until we figure this out, let's not try to do this.

So that's just a short way of saying, we have to do some more work with CDN. So is there anything we can do on our end to make this a little better? Well, let's quickly go into our code, because the answer is, yes, maybe. Depending on your philosophy.

So we have that code here that I mentioned. So we have similar code here. And what we're seeing is a link to the HTML5 JavaScript shim. And this is being posted on one of Google's servers. And so what we're going to do is we're going to remove this absolute URL. And we're going to replace it with a local URL, or the local JavaScript file.

That JavaScript file is sitting here in your lesson folder. And so we're simply going to replace this with that relative link. It just looks like this. Go to `html5.js` and use that. So again, I'm not going to take a huge position on one side or the other. There's still a lot of debate on this. But in the end, you can choose based on your organization or the conclusions that you come to.

Now there are also a couple of other concepts just to talk about. One of these is the future performance. So there's an API being developed called the Network Information API. And the job here of this API is to detect whether or not a user's connection is slow or not. And if it is slow, then there are things that we might be able to do.

So let's say you've got a slideshow. And within that slideshow, you only want to serve low resolution images if the connection is slow. Well, that's something that the network information API is planning to do. We can't quite use it reliably yet. It's not fully robust. Hopefully, shortly, we should be able to.

Something that is also new and can be used is conditional loading. Now this concept was first brought up by Jeremy Keith. And there's a great article here at [24ways.org](http://24ways.org), where he talks about conditional loading. And the way that this works is as follows: You only load items on a page as needed.

And to give you an illustration of that, let's go to the Boston Globe site. And here, we'll click on the Weather link. Now we can imagine that someone might come to the site and never have to click on the Weather link. But assuming they do, what pops up? Well, we get five days of weather here.

So conditional loading says, we don't actually load the contents inside that Weather section unless we need it. We don't want to penalize people who never click on the Weather link. And that's what we would have to do in a more traditional setup.

So this uses some Ajax here. I'm not going to get too much into the details. Again, if you want to look at those details, here is a link to an article that talks about that by our friend Scott Jehl. And if that name seems familiar, this is the same person who did `picturefill`. And so in many ways, you have conditional loading happening right now on your portfolio page with images. Go ahead and give yourself a small trophy, a little pat on the back!

Cool. So having said that, we are at the end of optimization. And we still have a couple things left to cover. So the next lesson, we talk about Grid Systems. And then the final lesson, we talk about Responsive Workflow. Until we get there, however, you've got some homework.

Assignment number 1 is always a short quiz. Assignment number 2, I'd like you to read this article, "Performance as Design." and here's the link to that article. Absorb it. Read about it. Talk about it in the Forum. It's good stuff.

And finally assignment, number 3. You need to put some of these techniques that we talked about into practice on your own portfolio page. Make sure you check your site in older browsers. Be sure to run the YSlow system on your site. And of course, if you haven't already, you're really going to have to focus and optimize images. And that's it. I'll see you in the Forum. And I'll see you in the next session.