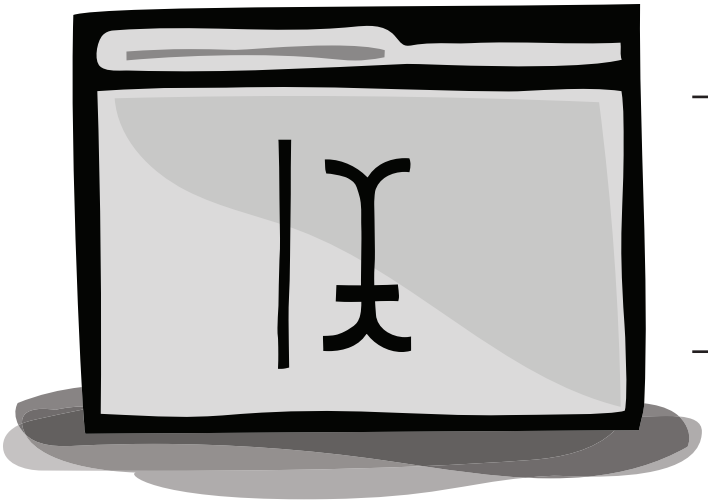


GYMNASIUM

INTRODUCTION TO MODERN WEB DESIGN

Lesson 5 Handout

CSS is a Design Tool

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. CSS allows you to control the background styles of HTML elements. Background styles can be applied to an entire page or just a single element such as a heading. You can add color, images, and gradients to a background with the proper syntax. You can also add multiple background images if you choose and control the positioning of those images as needed.
2. Borders in CSS can be applied to any side or all four sides of a styled element (border-top, border-right, border-bottom, and border-left). You can control the color, width, and style of these borders as well (border-color, border-width, and border-style). The border-radius property allows you to adjust the corners of a border in order to create rounded effects.
3. CSS allows you to generate or inject content into your design elements using the ::before and ::after selectors. It is important to note that generated CSS content should be used for design accents; important content should generally be formatted as HTML.
4. One of the primary layout tools a front-end developer needs to understand is CSS floats, which can be applied to one or more elements on a page. When you float an element, it removes it from the normal flow and pushes it either the left or right side.
5. When an element is “floated,” the default behavior is for other content such as text to wrap around it. If you do not want content to wrap around the float then you must use the CSS clear property, which allows you to prevent elements from being positioned to either the right or left side of the float.
6. CSS has a number of properties relating to positioning of elements including static, relative, absolute and fixed. The default value for the position property is static, which will be used the majority of the time; the other positioning properties have specific use cases aimed at creating unique effects within your layout.
7. CSS allows you to style content for specific media types—such as print—through the use of the @ media rule. When the @media rule is used to style content for different types of screens or devices (such as phones, projectors, or TVs) we refer to them as “media queries.” You can add multiple media queries to your stylesheets in order to create content that adapts and responds to certain properties of a screen or device; for example, if you wanted to format content specifically for the landscape mode of a phone, media queries provides the syntax to do so.

8. The CSS transform property allows you to manipulate an HTML element in two dimensions or three dimensions. Some of the forms of manipulation include scaling, skewing, rotating, and translating, which allows you to move the element in any one of the x, y, or z axes.
9. CSS transitions allow you to animate property values within an HTML element as they move from one value to another. This is achieved using the transition property, which allows you to set various characteristics of the transition including duration and many others.
10. In addition to CSS transitions, which are essentially basic animations, you have the ability to create more complex animations using the @keyframes rule. This feature allows you to control the various steps in a CSS animation sequence by establishing one or more keyframes over time along your animation sequence.

ASSIGNMENTS

1. Quiz
2. Enhance your design using backgrounds, borders, and generated content. We recommend starting with backgrounds and borders in order to separate your content and adjust how your eye is drawn across the page using those design tools. Some questions to ask yourself as you experiment with these properties:
   - How do solid color backgrounds compare to subtle gradients in terms of drawing your eye?
   - Knowing how you make a triangle, could you guess on how to make a circle out of an element?
   - Are there places in your design where it makes sense to use generated text or images?
   - When you're done with your enhancements, publish your work on GitHub, post a link in the forum, and post some of your observations of working with these different design tools.
3. Begin exploring the use of CSS media queries as a method for creating layouts for different size screens. To begin, shrink your browser width to roughly 320 pixels and adjust your layout to provide a good reading experience. Then, slowly increase the width of your browser until the design starts to look not-so-great, and use a media query to adjust the layout. Repeat this process at least once, if not twice, until you have a couple of different layouts for your design. Some questions to answer at the end of this process:
   - How many major and minor breakpoints did you end up with?
   - What did you find challenging about this process?
   - Publish your work on GitHub and post a link to it in the forum, along with your observations and answers to these questions.
4. Look for ways to include transforms and transitions in your site. For example, see what happens when you transition the width property for every element on the page as you resize the browser. Post your findings and what you've done in the forum.

INTRODUCTION

(Note: This is an edited transcript of the Modern 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.)

In this chapter, we're going to be discussing basic backgrounds, as part of an overall discussion of CSS as a design tool. Now this should look pretty familiar. We've used this a couple of times, just setting the background to red. But when we're setting a background of red, what we're actually doing is setting multiple properties at once because background is actually a shorthand for several properties. And what it's a shorthand for is background color, background image, background position, and background repeat.

Now, in the case of setting background red, we're leaving all of the background image, background position, and background repeat values to their defaults. But we could set each of those individually, if we wanted to, saying red for the background color, url(photo.jpg) for the background image.

Now, URL is a special function within CSS that allows us to point to a URL. That could be a fully-qualified URL, or it could be a local URL, a local relative, like you see here. It could be based on the site root. However it is, whatever you have as a valid URL will work in the URL function there. Now, then we have the background position, and finally, the background repeat. Background position sets what the position of the background image is with respect to the containing box. And then background repeat sets whether it should repeat at all, like a tiled background, or whether it should only repeat along one axis, or neither axis, et cetera.

```
p {
  background: red
              /* background-image */
              /* background-position */
              /* background-repeat */;
  width: 200px;
  height: 100px;
}
```

There are a bunch of other properties that are rolled up into this shorthand.

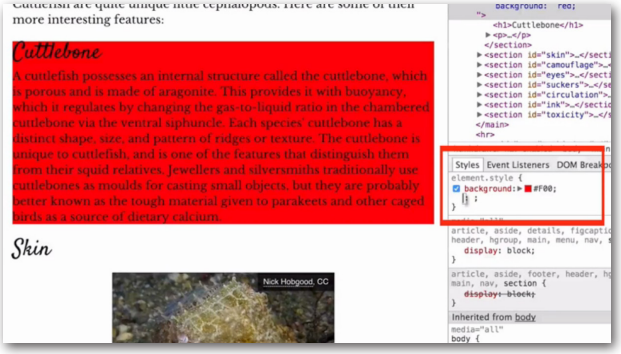
Incidentally, background position is also shorthand for background-position-x and background-position-y, which handles the positioning of the background image, with respect to the x- and y-axis—x-axis being the horizontal, y-axis being the vertical. So it should go in that order when you use background position, or when you use it within the background shorthand.

```
p {
  background-color: red;
  background-image: url(photo.jpg);
  background-position-x: center;
  background-position-y: top;
  background-repeat: no-repeat;
  width: 200px;
  height: 100px;
}
```

Incidentally, background-position is also shorthand.

Now, most browsers are smart enough that if you were to reverse the keywords and say, top and center for background position instead of center top, they will sort that out for you and assign them in the right direction.

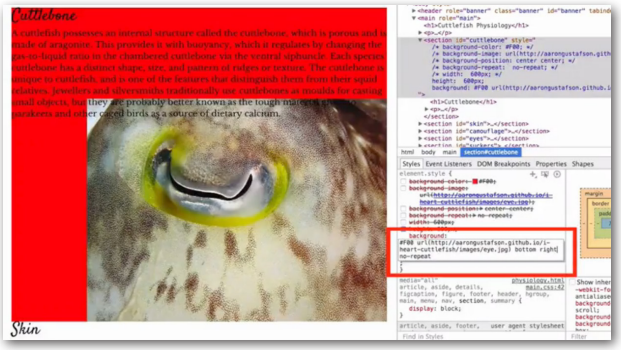
Where it becomes a problem is if you use unit values, such as percentages or pixels or what have you, as values for the background positioning, because those will have to follow the horizontal and then vertical values for those.



Now let's jump into the browser and take a look at how to use background. So if we click on this section element, I can go ahead and set a background of red on this element, and then I can set—I can change that back to being a background color, so I'm not using the shorthand. Let's do that. And then I can set a background image. And I earlier copied a URL for the eye image for my cuttlefish site. So I can drop that in there.

And then I can say that I want the background position to be centered both horizontally and vertically. And then, let's say that I do not want the background to repeat. Now you can kind of see a little bit of red on both sides there, where the image has been chopped off. But if I enlarge the size of the element here to 600 by 600 pixels, you can see that the image actually is centered both vertically and horizontally.

Now if I turn all of that off, I can actually use the shorthand for all of those values. And you see it does exactly the same thing. Now let's play around a little bit with the positioning. If I set it to top left, enlarge the browser here. If I set it to top left, you see that the image appears in the top left, as you expect. Bottom left, same thing. Bottom right, same thing.



And you can see that I'm not using the keywords in their right order. I'm not doing horizontal and then vertical, I'm doing vertical and horizontal and the browser is figuring it out. But if I set zero zero and I start adjusting them, you'll see that 20 pixels is doing the horizontal, not the vertical. And then, if I do 20 pixels again, it does it 20 pixel offset from that left hand and top edge.

If I switch them to percentages, the browser actually does something really interesting, where it divides up the percentage across the element with respect to the image. So if I set it to 20%, it's going to be 20% across. But if I set it to 100%, it's actually going to be all the way across, but not over. So it doesn't just take the left edge. The position actually adjusts as you move across the element. The actual offset moves across the image as well, so that you never kick the image out of the page.

And here, you see I can adjust the repeat to allow it to repeat in just one direction. Repeat along the x-axis. And I can continue adjusting even above 100%. Or I could get ridiculously silly and go really low. And you see it just keeps tiling along that horizontal axis, which is pretty neat.

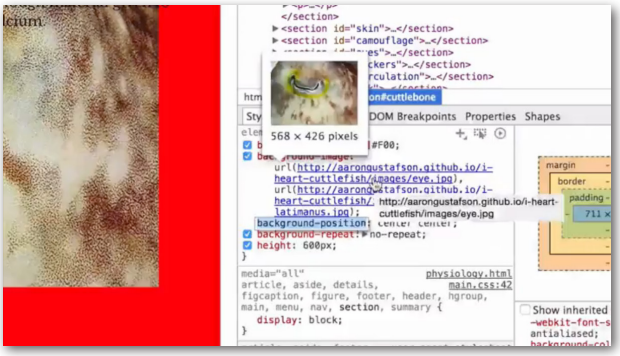
So in this chapter, we covered basic backgrounds and went through how the background properties work. In the next chapter, we're actually going to dive into more advanced backgrounds and look at how we can adjust the sizing of background images and other fun things like that. So hopefully I'll see you in a few minutes.

ADVANCED BACKGROUNDS

Welcome back to Chapter 2. In this chapter, we're going to be talking about some advanced background approaches using CSS.

So the first thing I want to talk to you about is multiple backgrounds. So as a relatively new addition to CSS, we've given the ability to have multiple backgrounds assigned to a given element.

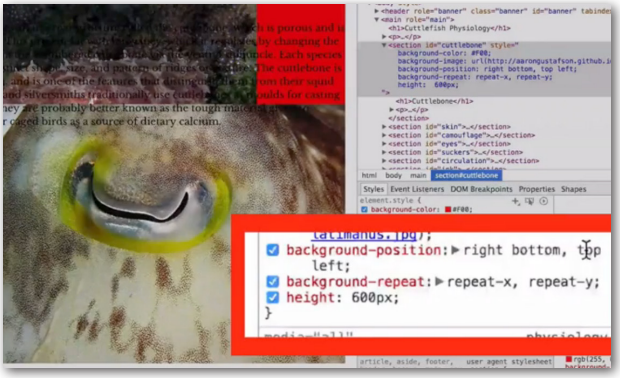
And the way that this works is that you use commas to separate the multiple background properties and then the backgrounds stack behind one another. That's really important to remember. They stack up behind one another.



So let's jump over to the browser here and take a look at the image that we had before with the eye. And I'm going to go ahead and drop in another image here. Adding a comma inside the background image property and adding another cuttlefish image, which you can see if I mouse over here.

But you're not seeing it because we're all position center, so one's on top of the other. If I adjust it so that one is in the bottom right and the other is in the top left, now you can see both of those images.

Now, each of the individual background properties can be adjusted with commas as well to allow for repetition, let's say. So here, the first one is not repeating but the second one is. Or I could have the second one only repeat along one axis. So the vertical axis, repeat-y. Maybe I'll change the first one to repeat along the x-axis.



So now, we have each one repeating in a different direction. And then I could adjust the background positions on those to maybe make them overlap in an interesting way. I don't know. It looks a little silly.

But there's a lot of power that we have with these controls to be able to adjust how images fit together as backgrounds on a particular element, which is pretty cool.

So multiple backgrounds can be defined using the background shorthand. Here you see we've got the, in orange, background. We've got one url to Sepia-latimanus.jpg and it's in the top-left with no repetition. So 0 0, no repetition.

The second one after the comma is URL images/eye.jpg. It's 100% 100%, so it's going to show up in the bottom right-hand corner. And it's not going to repeat.

And then the last one we have the color red. URL for images/tentacles.jpg at 50% 50%, so it's going to be dead center. And it's not going to repeat either.

Now, it's important to note, as you see here in green, if you need to have a fallback background image for browsers that don't support multiple backgrounds, it's important to have one without commas in it. Another best practice with regard to this is to only assign the background color in the last background. Because, as you remember, the backgrounds stack behind one another.

```
p {
  background: url(images/Sepia-latimanus.jpg) 0 0 no-repeat;
  background: url(images/Sepia-latimanus.jpg) 0 0 no-repeat,
              url(images/eye.jpg) 100% 100% no-repeat,
              #F00 url(images/tentacles.jpg) 50% 50% no-repeat;
}
```

Multiple backgrounds can be defined in the background shorthand.

So if you were to put the color on the first background, anything outside of that background image would have that color. So the color would be overlaying any of the images that would sit behind that layer. So just something to keep in mind. Good idea to have the color on the last one and to have your background—have a fallback for browsers that don't support multiple background images.

Now, another property that can be really useful in kind of advanced background manipulation is background-clip, which allows you to limit the amount of the image that is showing based on properties of the element itself. So if I jump back over to the eye again, I'm going to take off the height on it. But I'm going to go ahead and set a border of 10 pixels and dashed, so that's going to be black. And then I'm going to add some padding to it as well just so that we can get a little bit of separation from the content to the border.

Now, using background-clip, I can adjust where that image sits. So you see right now, it's outside the border. If I choose content-box, it clips the image and the background color to the size of the actual content itself.

And then inherit and initial are both border box as well. And then padding box is another option. And here you see the image is actually tucked inside of the border.

So there are a lot of options for us to tweak how the background is displayed with respect to the content.

Now, background-origin lets you adjust the x and y offset of the origin of the background. So this is another one that's probably best described in the browser.

So here again, we've got our example. And I'm going to go ahead and set the image to be in the top-left corner. And here you can see it sitting just inside the border.

And if I turn off the background-clip, you notice that the color extends outside but the image doesn't. This is the way default images layout. But if I set the background-image to border-box, now the image sits outside the border as well. And then if I do content-box, the image moves into the content.

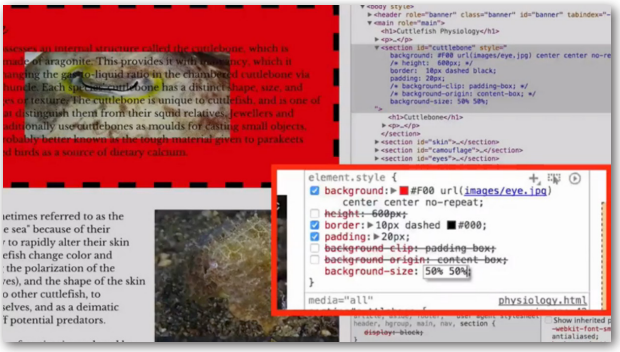
And then padding-box—it moves inside of the padding. But notice that it still overflows the border. I would have to adjust the background-clip as well in order to keep the image tucked inside of the border. So that's pretty interesting.

The last property I want to run you through is the background-sizing property. And this actually lets you tweak the size of a background. Usually this is used with respect to a background image specifically.

So let me jump over here again into the browser. And I am going to set this back to the center. I'm going to leave some of this other stuff on.

I'm going to jump down here and say background-size 20px. And you'll see the image gets 20 pixels wide. Tiny.

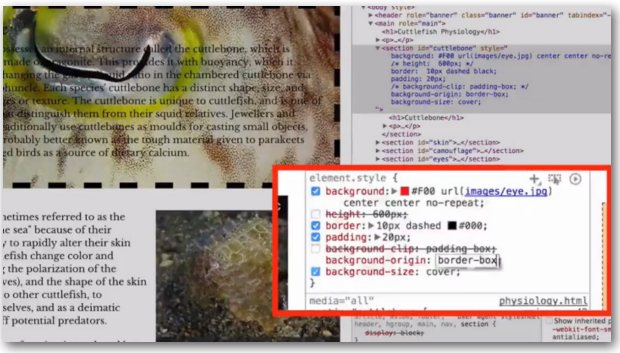
If I say 20px 20px, the image is actually stretched in both directions to be 20 pixels wide. Here I'm just making it taller, making it wider. And you can see the image is distorted when you do that.



You can set specific percentage sizes, so 50% 50%. And those sizes are with respect to the element, not to the image, which is important to recognize. OK.

And if I just set one value, it does it to the width of the image. If I set 100% 100%, it covers the whole thing. I can adjust the size also to use the keyword cover, which allows it to cover the entire element automatically without having to do 100% 100%. I can adjust the background origin to being the border-box so that it pops outside.

And now we have a full-column image that will always stretch to be the right size. But the proportions are maintained. So part of the image will be clipped off on either side, but it will fit appropriately.



Another keyword I can use is contain, which will make sure that the entire image is always visible. Cover tends to be the one we see a lot on the web currently. And that's because it's used for a lot of big header shots that you want to make sure are always full width in the browser and that you never have any empty space around the image.

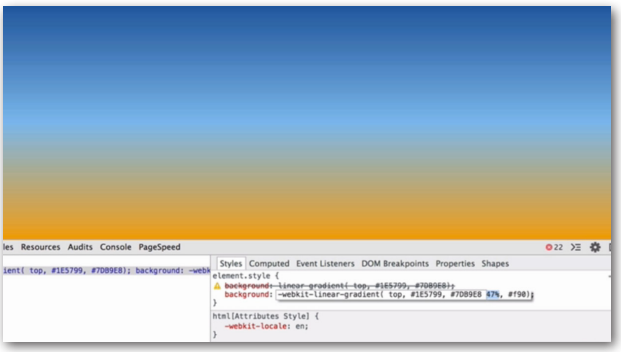
So the last thing to know is that each of these can support multiple backgrounds as well. So if you wanted to adjust each one specifically to the image that you're using in your multiple background, you just put commas in the property values in between the different ones that you want to use and want to assign to each of those backgrounds—which is kind of cool.

All right. In this chapter, we looked at some advanced background techniques. And in the next chapter, we're going to look at generated gradient backgrounds, which are pretty interesting. I'll see you in a few minutes.

GENERATED BACKGROUNDS

All right, welcome back. In this chapter, we're going to talk about generated backgrounds. Now, what do I mean by generated backgrounds? Well, what I mean by generated backgrounds is we can actually use CSS to generate in a gradient or colors or what have you, in order to make things that maybe we would traditionally do with images.

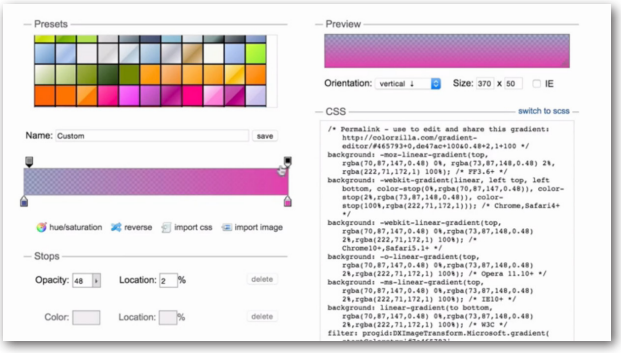
So let me show you quick example here. Here we have a basic linear gradient, which I tried to type in, but because it's still a vendor-prefixed property, I need to do WebKit linear gradient here in Chrome. But this sets the gradient as traveling from the top to the bottom and being two color stops, and I just tweaked it to being a third color stop.



And I can adjust where those color stops fall in terms of the gradient. Here you see me adjusting it lower in the percentage. I can raise it up in order to tune the linear gradient to exactly what I'm looking for.

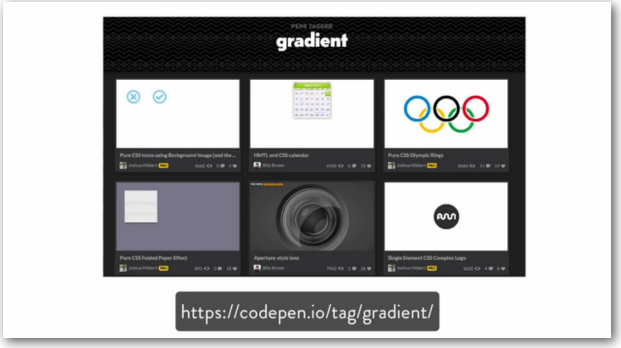
Now, this syntax is kind of obtuse to look at. I'll be the first to admit that. So what I've really found to be a super helpful tool is ColorZilla's gradient editor. And with the gradient editor, we can actually go in and choose from a bunch of predefined gradients, or we can dive in and take one of them as a starting point and then tune the different pieces, adding and removing different color stops, adjusting where the colors fall.

We can even do things like adjust what the actual color is in order to tune it to what we want. Maybe I want something of a darker blue. And then I can even adjust the alpha transparency of that color in order to get just the look that I want for the design that I'm trying to achieve.



Now, on the right-hand side, what you'll see is that I have the ability to control orientation and the size of the gradient. And all throughout, the editor is generating the CSS for me, and it gives me of the vendor-prefixed versions from the Mozilla version, the two different WebKit versions for old WebKit and new WebKit, the Opera version, the Internet Explorer version, and then finally the standards version. It even throws in an extra filter version which is used for IE 6 through IE 9, which is pretty awesome.

This is just a taste of what you can do with gradients. If you look on CodePen, there are a ton of beautiful examples of really interesting things that people have done with gradients.



And you can create some really interesting effects with them all without having to resort to using actual image elements, which is fantastic. It saves a lot of download resources for your users.

So in this very brief chapter, we've discussed background gradients. And in the next chapter, we're going to tuck in to borders. I'll catch you in a few minutes.

BORDERS

Welcome back to our continued discussion of CSS as a design tool. In this chapter, we'll be talking about borders.

We just looked at a bunch of different background options. And backgrounds can be fairly complicated. Borders, by contrast, are far less complicated.

So in a very simple way, we can say we want to have a border that is one pixel and solid. And remember that the border will automatically pick up the foreground color of the text and apply that to the border.

But border, like background, is actually shorthand for multiple properties—border-style, border-width, and border-color. Now, that original rule of border 1px solid could be rewritten as border-style solid: border-width 1px. The border color, as I mentioned, is automatically the foreground color.

You can set the border color explicitly. So you would have border-style, border-width, and border-color. And just to repeat, here is the shorthand again: 1px solid red.

Now, interestingly, each of these is also a shorthand for another property because we have four different sides to apply a border to. So when we're saying border-style solid, what we're saying is border-style solid solid solid. When we're saying border-width, we're saying border-width 1px /* 1px 1px 1px */. And when we're saying border-color red, we're saying border color red /* red red red */.

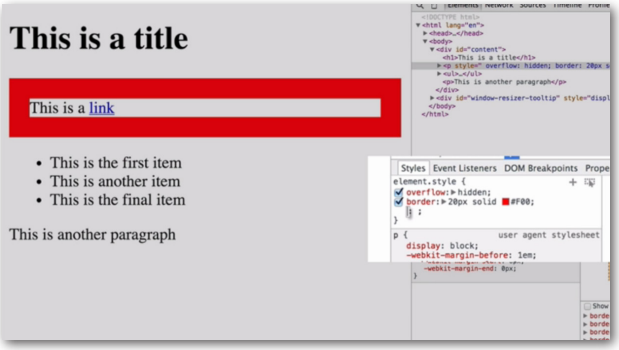
So this works very much like with padding. And we can set individual properties if we wanted to. If we wanted to say that the top was going to be a solid border. The right was going to be a double border, so two lines. The bottom was going to be a dotted border. And the left was going to be a dash border. And they were going to each have different widths. We could do all of that by setting the individual properties.

We could also roll these up into their own shorthand as border-top, border-right, border-bottom, and border-left. So there's a lot of options with how you want to put your borders together.

In most cases, you'll probably have a single border being applied to the entire element. And then maybe you'll change that border for one specific side if you wanted to make that one side less consistent than the other sides.

Now, interestingly, you can use borders to make shapes, like a triangle. So let me jump over to the browser here. And just kind of demonstrate how borders can be used.

I'm just going to take this paragraph as an example. I probably wouldn't in reality do this in a paragraph. I might use it as generated content. But I want to go ahead and show that to you here, just since we've got this demo document.

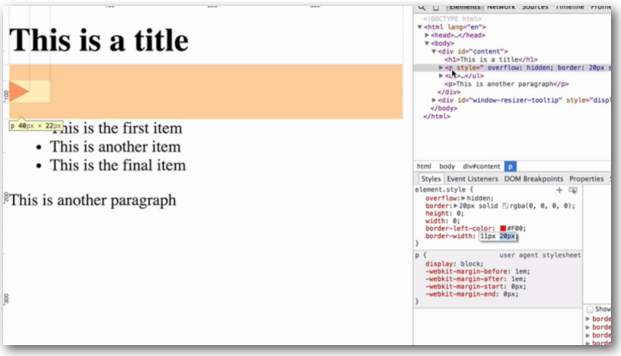


So I'm going to set a border of 20 pixels solid in red all the way around. And then I'm going to set the height and width to zero. And I set the overflow to hidden so the content will not be displayed outside of that zero pixel, zero pixel box. So it makes it a 40 pixel square. All right.

Now, if I change the color to transparent, the borders disappear but they're still there. I can then adjust the border color on one side to be the color red.

So if I set the border-right color to red, we have a left-facing arrow. If I adjust that border color to be border-left color red, then I get a right-facing arrow. If I make the border-top, I've got a downward-facing arrow. And border-bottom, I can make it and make it face upwards.

So that's kind of cool. And I can adjust the different sizes as well in order to tweak the shape of the arrow.



So all of that is because of the way the borders get composited by a browser. It creates that wedge shape. But if I mouse over it, you can see that the actual borders are there. They're just transparent. So borders are pretty cool.

Now, another thing that we have with borders is that sometimes we don't want them to be a square, right? So in this instance, we can actually use border-radius to round those sharpened corners.

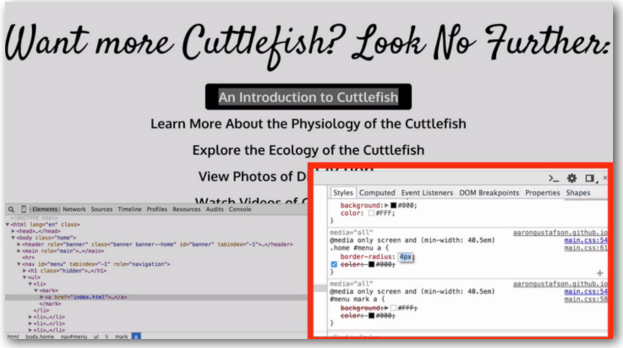
And there are several longhand properties that we can use to do that. Border-top-left-radius, border-bottom-right, and so on and so forth. That's the longhand. Or we could simply say border-radius: 10px. And then I would set it to all four corners.

And like I gave you the mnemonic "trouble" (TRBL) for Top, Right, Bottom, Left for assigning padding, the same goes for borders and margins. With border-radius, it starts in the upper left-hand corner and then goes clockwise as well.

So it's the top left, then top right, then bottom right, then bottom left as it moves around. So you could separate these with spaces as well.

Now let's jump over into the browser and look at the cuttlefish website that I've put together. And down at the bottom of the page, I'm using border-radius on these links. I'm going to enlarge the text just so you can see it a little bit better.

And then—all right, let's inspect this.



And then what we'll see is that I've got a border-radius of 5px set on there. I can increase that. And you see that the actual thing becomes more lozenge-shaped, right?

I can shrink it and then the curve gets smaller. So at a certain point it just starts looking round. OK.

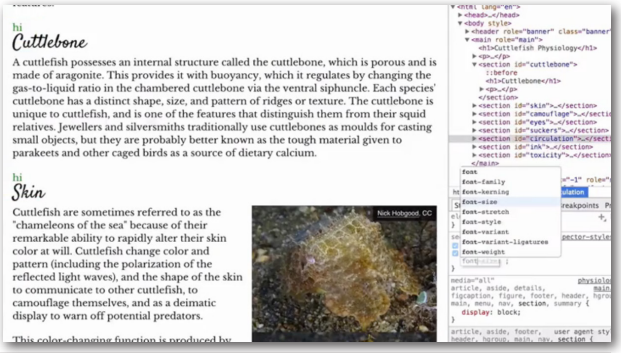
Now, there's a newer syntax that allows me to put a slash in there and actually adjust the two different vertices of the curve separately. So that allows me to create some interesting shapes that aren't perfectly round. You won't use that quite as frequently but it is there as an option if you do want to create some interesting shapes.

So in this chapter, we talked about borders and we talked about the border-radius property. In the next section, we're going to talk about generated-content, which I mentioned a little bit earlier with respect to the triangle. I'll see you in a few minutes.

GENERATED CONTENT

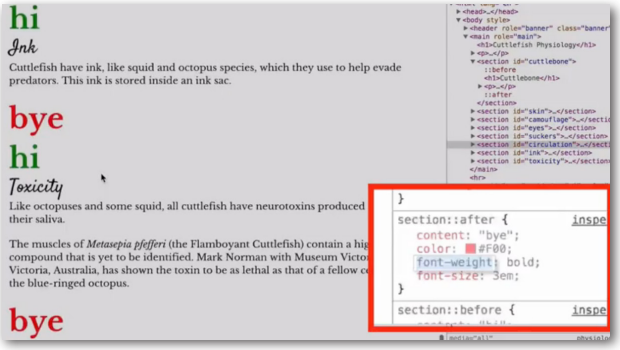
Welcome back. In this chapter, we're going to be discussing what's called generated content. Now, generated content uses a specific form of selector that is a pseudo-element selector. So two colons, although most browsers will allow you to do just a single set of colons, and ::before and ::after as well.

So ::before and ::after inject content into the page from CSS. As a general note, generated content is great for design accents that might go away or change, such as having colons after labels in a form or that sort of thing.



So to give you a demo of how generated content works, let's hop over to the browser here. And before every section, I'm going to inject some content. So if I say section colon before, and then I say content, "hi"—oops, let me put this in quotes—"hi," now you'll see the words "hi" up here before each section. I'm going to make these green, and a little bit larger and bolder, so that you can see them a bit more.

All right, so there's "hi." So that comes before each of the sections. And then if I go ahead and make another one after — do section colon after — and then give it the same styles. I'm going to say it's "bye." And the color, I'll make it red. In this case, it's injected the "hi" before and the word "bye" after each of the sections in the page.



Now, it's worth noting that the content that you've injected in here is not actually selectable using your cursor, but it does appear in the page, which is why it should be reserved purely for presentational content.

Now, a place where it could be really useful to inject this would be in the print medium, but I'm going to show it to you in the browser here. But I could say for every link that I want to go ahead and inject the URL afterwards. So I can do that by saying content, and then I'm going to do a parentheses here, and then I'm going to use the attribute function in CSS to actually pull the value of the href attribute out and inject that into the page as well.

So here I've gone ahead and done that. And you see now that after skin, it has, in parentheses, #skin, because it's an anchor to the skin section in this page. And this has happened for every link on the page. So that's probably not what I want to have happen, because maybe I don't want to have that happen on the figures and that sort of stuff. And you see it's happening in the navigation as well.

Hmm, so what if we went ahead and said main a::after. Now it's not affecting the nav anymore. And then I could say, any figures inside of the main, they're a—I'll just say figure a, after content is empty quotes.

So now we've basically turned off the generated content for those figure images, but we still have the generated content in the main content. So this could be used to generate in content -like warning iconography or any sort of iconography that you want to put into the page using a URL just like we did for our background, and that will inject that image into the page, and we can position it and do all sorts of other things to adjust that.

The one thing that I should note with injecting an image into the page using generated content is that you cannot change the size of it. So make sure you're happy with the size before you go ahead and do that.

So that's it for generated content. I'll let you play around with that a little bit more on your own. In the next section, we're going to talk about margins and how exactly those work.

But before you jump over there, I'd like you to go ahead and do Assignment Number 11, which is to enhance your design using backgrounds, borders, and generated content. So the steps that I think you'd probably, most likely want to go through in order to do this are to use backgrounds and borders to begin to separate your content and adjust how your eye is drawn across the page using those design tools.

How do solid-color backgrounds compare to subtle gradients in terms of drawing your eye? Knowing how you make a triangle, could you guess on how to make a circle out of an element? That could be pretty interesting as well.

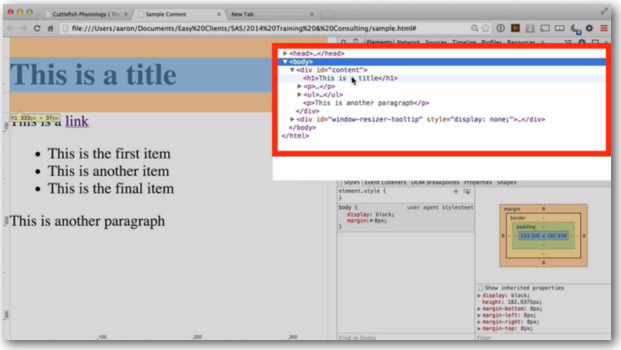
Are there places in your design where it makes sense to use generated text or images?

And then when you're done kind of playing around with this stuff and thinking about it, go ahead and publish your work on GitHub, post a link in the forum, and post some of your observations of working with these different design tools. We'll see you in a few minutes to talk about margins.

MARGINS

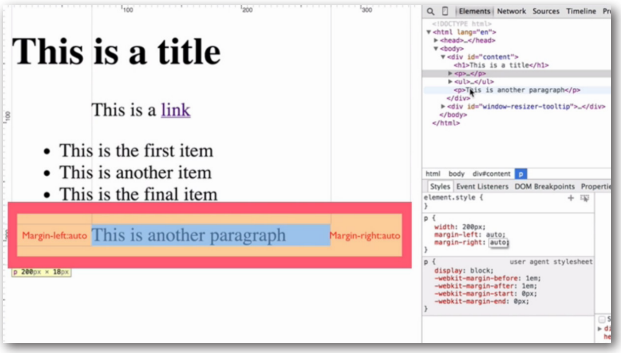
Welcome back. In this chapter, we're going to be discussing margins. Now margins govern the space between elements. So let's take a look at this in the browser.

If I mouse around this document, you can see that the body has got some margins on it there. Content does not. H1 has margin above and below. Moving down into the paragraph, above and below. But you'll notice that the margin on the bottom of the H1 actually overlaps the margin on the paragraph, and the larger one will win out of that.



Now when margins collapse, the largest one always remains. And tipping over to the browser again, let's go ahead and set a size on this paragraph here. Here you see the paragraph margins, but I'm going to go ahead and adjust those. So if I set a width of 200 pixels on the paragraph, you see the margin auto expands to fill the rest of the line. That's an automatic margin.

Now, if I tweak the margin and set the margin left to Auto, you'll see that the margin automatically fills on the left-hand side. If I set the right to Auto, then it automatically centers the element, autofilling the space, dividing it equally on either side of the element. So this is a great way to do some horizontal centering.



So margins can be used to autofill space on either side. So that in a nut shell is margins. And in the next chapter, we're going to talk about floats and clearing, which when combined with margins end up becoming a really useful tool for laying out pages.

FLOATS

Welcome back. In our continuation of the discussion of CSS as a design tool, we're going to tackle what are called floats. So the float property is one of the tools we use to layout content on the page. When you float an element, it removes it from the normal flow and pushes it to a side—either the left side or the right side. And when you do so, you could either refer to it as a peninsula (or an island) with the normal flow content flowing around it.

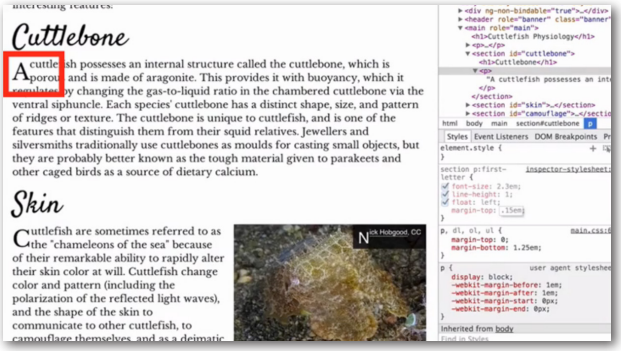
So if I were to say something like p, first of type, double colon, first letter, this would give me a drop cap that is four times the normal size. It's bold. And it would be floated to the left. And then the normal content would flow around it.

```
p:first-of-type::first-letter {
  font-size: 4em;
  font-weight: bold;
  float: left;
}
```

Floating creates a peninsula (or island) and the "normal flow" content flows around it.

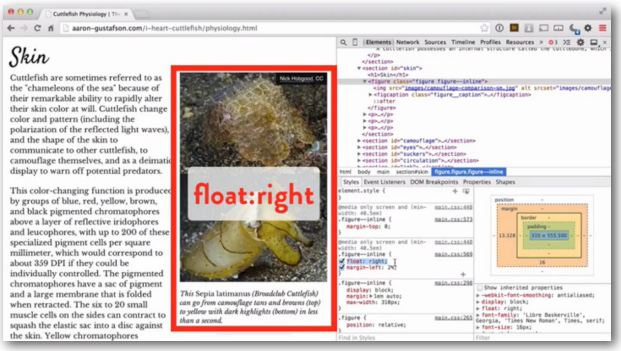
Let's look at some floats in the design of my cuttlefish site. So if I say for every section, the first letter of the paragraph is going to get a font size of 2em. And then let's say I wanted to adjust this line height just so it doesn't bump things out weirdly. But then I'm going to set it to float to the left. And now you see the text is wrapping around that letter. If I adjust the font size, make it a little bit bigger. It's about two lines high. We can make it huge, and you see the content just flows around it.

Now if I set a margin, I can adjust the top of the text to push it down a little bit. And then maybe I want to set a margin on the right hand side to push that text out just like—and that looks a little bit better.

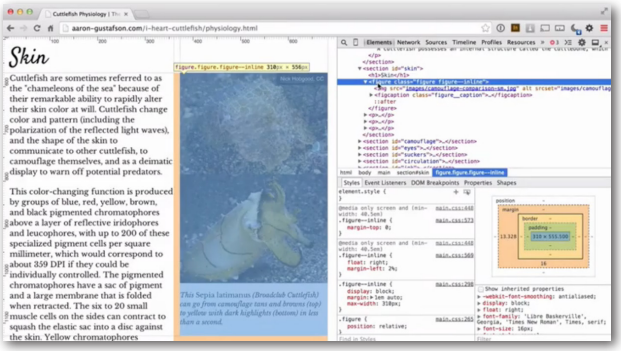


Now oftentimes when you're working with floats, you need to be able to clear a float in order to stop content from wrapping around it. This is where the clear property comes in. Now let's take a look at ths in the browser, because it probably is the sort of thing that you need to see in order to really grasp.

So here we have a section. And inside of that section we have a figure. This figure is floated to the right. Now if I were to shrink the size of the paragraph—or of the section rather—remove some paragraphs. Let's go ahead and do that. Go ahead and delete a couple nodes here so that it shortens up. Now you see camouflage is actually wrapping around that floated text still. So it remains in the normal flow.



Now if I were to go into the section here and say, I want all sections to clear right. Now the camouflage section no longer wraps around that floated content.



So an interesting thing that happens with floats is that if you float all the children of an element, the element itself collapses, because it has no elements inside of it that are holding it open. Nothing is in the normal flow. So let's take a look at that.

If we jump back over here to the browser and go ahead and add a background to this section here— so the background's red just so you can see how big it is—you see that the figure floats outside of it. Now I'm going to delete the heading, and I'm going to delete the paragraph. And you're going to see that the red just disappears. Look at that—no red visible at all. I'll add the paragraph back in and it appears again.

So what's happening is that there's no children that are not floated inside of that container. So that containing element, the section just collapses. So it has no height whatsoever.

```
.layout-container::after {
  content: " ";
  display: block;
  clear: both;
  width: 0;
  height: 0;
}
```

This is a handy way to ensure the parent (in this case .layout-container) is as tall as the tallest child.

GYMNASIUM *Introduction to Modern Web Design*

Now using generated content, which we talked about earlier in this lesson, we can actually generate in an empty space, set it to display block to clear on both sides, and give it a width and height of zero. So we're effectively inserting something that doesn't really exist visibly on the page, and then make that hold open the parent.

So if we've got the section, which here I have it highlighted. And you can see on the left-hand side there that the width is 667 pixels, but the height is zero pixels because it's not being held open. If I go ahead and add that generated content, you can see that the red already flowed to fill the entire space. So because there is that generated content after the figure, it can hold open that parent and make sure the parent aways contains any floated children inside of it. So that's pretty neat.

Now another trick you can use is to say on the ".layout-container—or on the section or whatever it is, overflow: hidden, which we saw earlier with the triangle example and some of the other stuff. But overflow: hidden has potentially unintended side effects in that if you try to move anything outside of that container using positioning—which we'll talk about in the next chapter—then all of a sudden, nothing appears.

```
.layout-container {
  overflow: hidden;
}
```

Sometimes you'll see this, but it is limiting in terms of what you can do with descendent elements.

GYMNASIUM *Introduction to Modern Web Design*

So overflow: hidden will expand to contain any floated children. But you do run into some risks with trying to move stuff outside of that parent container. For that reason, generated content is the best practice for autoexpanding a parent to contain any floated children.

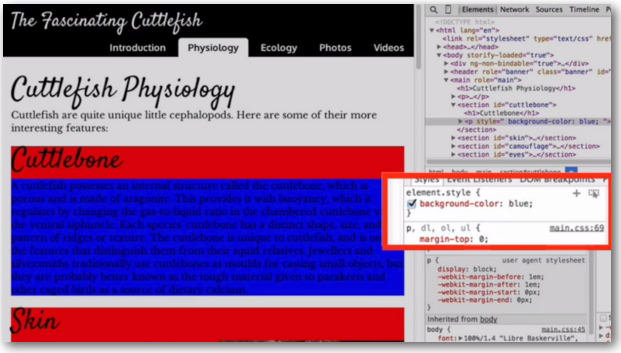
All right, so in this section, we talked about floats and clearing. And in the next chapter we're going to talk about positioning.

POSITIONING

Welcome back. In this chapter, we're going to be talking about positioning, which is another one of the properties that we use oftentimes for layout of our pages.

So the first thing to know is that every element is positioned static by default. So every element is technically positioned, but it's positioned statically, which means it's in the normal flow.

Now, elements with position relative remain in the normal flow just like static, but we can adjust their offset. So let's take a look at this in the browser.



If I go ahead and click on this section here, I'm going to change its color red so you see it. And then inside, I'm going to grab this paragraph. And I'm going to give it a background color of blue.

And then I'm going to position it—if I could spell—I'm going to position it relatively. And then I'm going to adjust one of its offsets, top 10px. So you see it's just shoved down 10 pixels. If I adjust that up, it moves it up.

You see, the parent container is still the height that it would be with that element statically positioned in there. We're just adjusting the offsets on one side or the other. So the paragraph still takes up the same amount of space.

I often refer to this as astral projection or an out-of-body experience because the element still takes up the amount of space that it took up initially but you're able to move it around with respect to that using the various offsets. And the offsets that we have available are top, left, bottom, and right.

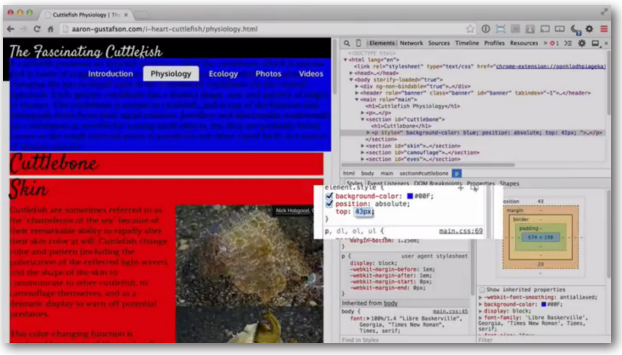
So offsets move non-statically positioned elements around the page. Relative offsets are moved with respect to the elements original position in the flow. So that's what we saw. The same space was taken up where the element existed before, but we're moving it around with respect to that original position.

Now, position absolute is slightly different. When we use position absolute, those elements are removed from the normal flow, which means their space is collapsed. Sort of like the floats, all right?

So let's take a look at this in the browser. I'm going to leave the blue, but I'm going to set position absolute. And look, it collapses down.

Now, the section that comes after skin actually slid up underneath the cuttlebone one. Now I can adjust the margin so you can see that the skin slid up underneath it. Interesting, right?

Absolute offsets are a little different than relative offsets in that they are calculated with respect to their positioning context. Now, what exactly does that mean? Let's kick it over to the browser so that I can show you.



If I say top 0, then its positioned with respect to the HTML element, which is the root element of the page. Notice that without a left offset, it keeps the left edge that it had before.

If I set the left to zero, it moves to the very left edge of the page. So that can actually be really useful to only set one offset, perhaps, so that it keeps the same alignment on the left edge of the text but maybe pops up to the top.

But you could set as many of these sides as you want to, which we'll look at in a moment.

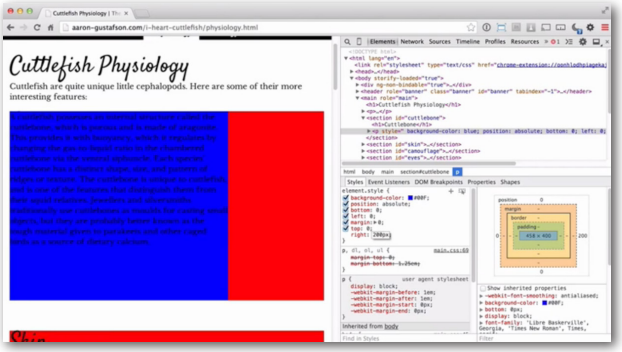
So the default-positioning context, as I mentioned, is the root element or the HTML. But the positioning context is actually established by positioning an ancestor element. So it could be a parent, could be a grandparent. By default, it's the HTML. But let's look at the browser again.

If we come back out here, we've got a position absolute and top left on that paragraph. But if we come down to the section and give that section a position that's other than static, now it's positioned with respect to that section. Obviously, because it's positioned absolute, it's taken out of the normal flow and so the element collapses. But we can make it larger in order to show you that that's where it's put.

Now I can adjust the offset. Say bottom left instead top left. Get a little space down here at the bottom so you can see it's offset slightly from the bottom there. There we go.

And that's because that paragraph has a margin bottom of 1.25 if you look at that next thing. If I set it to zero, then it sticks directly to the bottom of the parent element.

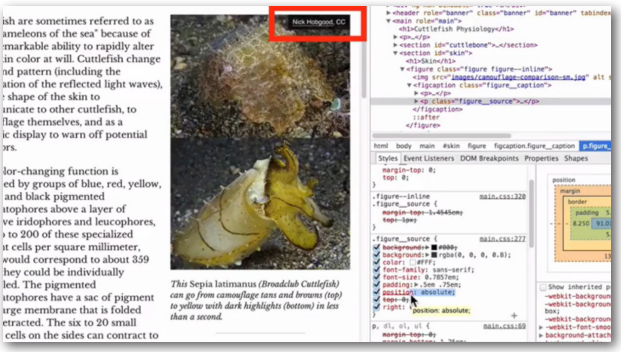
So I've got a bottom and I've got a left. What if I set a top? Hey, look. It covers the entire parent.



Same thing goes if I set a right. Now it will always be offset just that amount from whichever side I've decided. Some really neat layout effects that you can achieve using absolute positioning. You just need to keep in mind that absolute positioning does remove that element from the normal flow.

So let's look at another example of this in play. Here we have our figures on the cuttlefish page. And inside of that, I've got a position relative on the figure element. In this case, assigned specifically in a figure class.

And then I've got to figure__source, which is positioned absolutely. And it's positioned into the top right-hand corner. That's how we get that little black flag in the upper right-hand corner that, in the case, which says Nick Hopgood, CC.



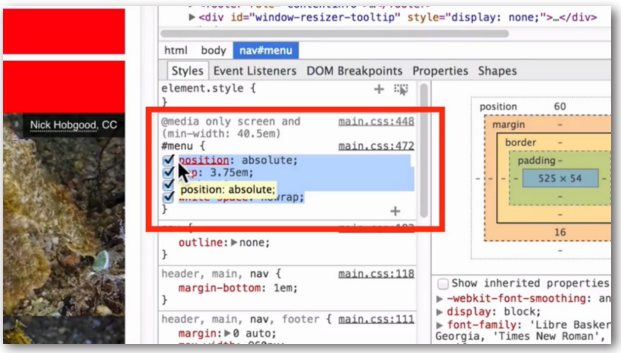
Now, positioned elements actually stack on top of one another. You probably noticed this a bit in the video that you saw earlier, where when we positioned that element absolutely with respect to the HTML, it slid up underneath the navigation. The navigation stacked on top of it.

So by default, stacking order is determined by source order. So as you move down the document, elements stack above one another.

So here we see that with the text of the navigation being on top of the positioned elements. I've positioned that absolutely so it stacks above another absolutely positioned, or relatively positioned, or what-have-you element on the page.

Now, that said, I can use the z-index property to adjust the stacking order. So let's go ahead and look back in here. And I've got our navigation. I want to take my paragraph and make sure that sits above the navigation.

So if I give it a z-index value of 1, the main nav doesn't have a z-index value, so now the paragraph sits above the navigation. So z-index can be a really useful way to adjust which elements sit on top of other elements.



Finally, the last type positioning I want to go over is position fixed. Position fixed is a lot like absolute. But the positioning context isn't the HTML element—it's actually the viewport. So it's what we're looking at the screen in. So let's actually look at that in the browser.

So if we have our absolute positioned paragraph up there at the top, you see it's on top of everything and it's stuck up at the top left-hand corner. But if I came in here and I said position fixed, it still sits up there. But now as I scroll, it's actually locked to the top of the page, which can be really useful.

So fixed elements will never scroll out of view. It's worth noting that on some older mobile devices, trying to fix an element to the bottom of the viewport can create some weird effects as you're trying to scroll the page because the browser's processor isn't quite as fast as a desktop. So it's constantly having to recalculate where that bottom edge is. So you can sometimes have lag or it just won't work at all.

So just something to keep in mind. Most modern mobile browsers can handle it. But it's just something to be aware of.

So in this chapter, we covered four different kinds of positioning. Position static, which is default. Position relative, which adjusts how an element is positioned with respect to its original position in the page. Position absolute, which adjusts where an element goes according to its positioning context, which is either an ancestor element that has been positioned or the HTML by default. And then finally, position fixed, which will lock an element to a particular place with respect to the viewport.

In the next chapter, we're going to talk about working with alternate kinds of media. But before you get there, I'd like you to go ahead and do Assignment Number 12.

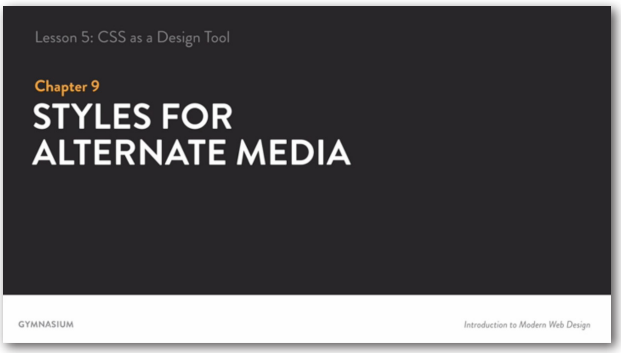
So adjust the layout of your page using margins, floats, and/or positioning to make it look the way you want. So are there elements that are currently taking up the full width of the page that might look better on one side or the other? Would increasing the white space between elements help in reading and understanding your content?

Look for opportunities to play with the different positioning types and create a layered design where elements sit on top of one another. When you're done, publish your work to GitHub and post a link to the forum along with your observations.

We'll see you in a few minutes.

STYLES FOR ALTERNATE MEDIA

Welcome back. So we've talked a lot about designing for the screen. But one of CSS's major powers is that you can actually use it for different media as well. So we can isolate specific CSS rules to a specific media type.

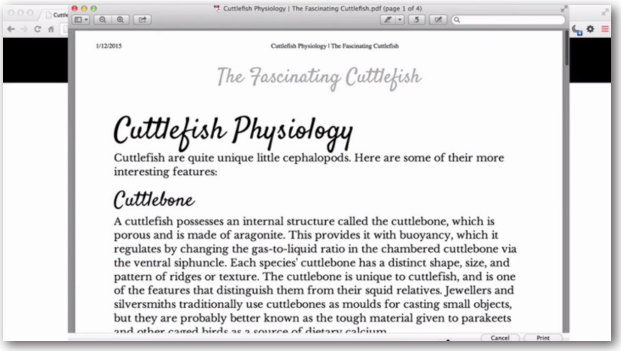


Now we may have seen this earlier in the HTML discussion, or in the CSS discussion of where to put your style sheets, so that we can assign a media attribute to a link tag. So in this case, I could say that I only want the print.css file to apply in the print medium. That's pretty cool. We can also do that within a stylesheet itself using an @ rule, @media print. And then only within the print context will the body size, in this case, be 14 pixels.

Or we can do it for a collection of rule sets inside a stylesheet using the @media block.

Hopping over to the browser, we can inspect an element. And here in the head, can see that it applies to @media all. And then if I go into that style sheet, I'm going to make this a little bigger so you can see it, if I scroll all the way down to the bottom. I have my @media print, which has all of the style rules that I'm going to use with respect to print. And I can set things like the page size. I can adjust the number of widows, all of that sort of stuff.

If I hit Print Preview, you can actually see the application of those styles to this page. So you see, I've used my generated content to go ahead and adjust the links to make sure that those appear afterwards. I've also done things like keeping the font size, getting rid of the navigation, getting rid of the images, that sort of stuff, with the exception of the actual text content. So that's pretty cool.



With @media rules and with the media attribute, we can actually assign multiple media that we want to apply something to. And we do that using a comma. So here you see @media print, refrigerator. And what's interesting about this is it looks a lot like the compound selectors that we can use in order to choose something to style.

```
@media print, refrigerator {

   body {
      font-size: 14pt;
   }

}
```

Note the comma between the two media assignments.

But it works a little bit differently, whereas with a compound selector, if any of the selectors weren't understood, the entire rule set would be ignored. With the @media block, if any of these are not understood, they're just ignored. So in this case, refrigerator doesn't exist as a standard media assignment. So browsers that come along won't know what to do with refrigerator.

So they just throw that value away. But they do understand @media print. So they keep that. So they won't throw away the entire @ rule. They just ignore the piece that they don't understand. So unlike in selectors, if one medium doesn't apply, the others still could.

All right, so this has been a basic overview of working with media. In the next section, we're going to talk about media queries, which take @media rules a step further and make them more capable of doing neat things like responsive design. I'll see you in a few minutes.

MEDIA QUERIES

All right. Welcome back.

So now we're going to build on what we learned about media assignment to talk about media queries. So what media queries do is they build on media assignment to allow you to test certain conditions.

So here's a simple example using link tags. So we have our basic CSS, and then we have our advanced CSS, and then our print CSS.

We have the media attribute for print on the third style sheet. But on the second one, we have this slightly more complicated media assignment: Media="only screen and (min-width 20em)".

So this is what's called a media query. And what it's looking for is only the screen medium and only to apply the styles that are within this advanced.css file when the browser width is at least 20em or 320 pixels, since an em is 16 pixels as the base font size on an HTML element.

```
…
<head>
   <meta charset="utf-8"/>
   <title>Page Title</title>
   <link rel="stylesheet" href="/c/basic.css">
   <link rel="stylesheet" href="/c/advanced.css"
         media="only screen and (min-width:20em)">
   <link rel="stylesheet" href="/c/print.css"
         media="print">
</head>
…
```

We can do it for an entire stylesheet in HTML using the media attribute.

Now, we can do the same thing inside of @media rules, saying only screen and (min-width:20em) in an @ rule. The only keyword says to only apply the rules in that specific medium. And it also works as a great signifier for using media.

Browsers that don't understand the only keyword will ignore that @media declaration or that media assignment in the media attribute and won't apply the rules at all. So this becomes a great way to deliver a normal stylesheet to all browsers. And then another stylesheet that's more advanced to only browsers that are media query using the only keyword as part of the media assignment.

So that's a great way to build something mobile first.

So when we talk about creating these different conditions, we're often talking about creating breakpoints. So major breakpoints refer to large layout changes that we're making to accommodate browser width most often.

You could do browser height. Or there's a variety of other factors we can test for which we'll show you in a minute. But mostly when we're talking about major breakpoints, we're talking about major shifts in the layout that we're making to accommodate larger and larger screens.

Minor breakpoints, or as they are sometimes called, tweakpoints, are small changes that we're making to individual elements within the layout.

Now, it's worth noting that there are new standard media queries. You'll often see certain examples of 320 pixels, or 20em, and that sort of stuff. They crop up every now and then. But the reality is that you should only be creating media queries that are appropriate to you.

Now, Stephen Hay, who's a designer based in the Netherlands, has a really great strategy that he talks about where you start with a small screen. And you start to enlarge the screen until it starts to look not so great. Then you insert a breakpoint.

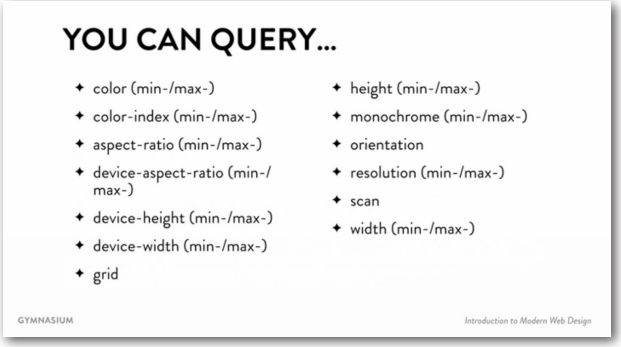
Then you do the same thing again. Rinse and repeat. Keep inserting breakpoints as you need them as defined by the content. So you're not defining your breakpoints based on certain devices.

As I mentioned, the only keyword can be used to filter out older browsers. And then you can also use and and not to further refine the query. So we saw this in “only screen and (min-width 20em).” So these rules would only apply on the screen medium when the viewport is 20em or larger.

We could say “not screen and (color).” So it negates the entire query. So the rules inside of this @media declaration would only apply in non-screen media that don’t support color.

Media queries, just like media assignment, can be compound as well. You could do something absolutely ridiculous like this, but please don’t. This is getting a little bit absurd.

But you can query a wide variety of different things, such as the color, the color index, the aspect ratio of your page. Information about the device, such as the device aspect ratio, device height, device width, whether it supports a grid or not. The height, whether it’s monochrome. The orientation of the browser, whether it’s portrait or landscape, the resolution, and most often, you’ll see the width.



Now, you might be wondering how exactly do you end up with min and max color and min and max monochrome and such? The color things are numeric because the value refers to the number of color bits. And resolution is in dots per pixel or dots per inch. Those are the values that you would use when testing resolution.

When referring to width within your media queries, ems are generally the best way to go. Let’s jump over to the browser and see what that looks like.

Now, if we zoom in over here and look at the media queries that are being assigned to the banner here, you see we’ve got media=“all.” and then we’ve got media=“all” with a min-width of 40em. We can use the mobile testing tools within Chrome to actually adjust how this is being displayed.

So I’m going to change it. Let’s pick—leave it as generic notebook. We can see it like this.

Let me go ahead and change it to a notebook with an high dot-per-inch screen. You can see it got a little bit tighter. I can adjust some of these things manually to shrink it down.

And what you see is that actually the styles are changing based on that. OK.

So if I went to a Kindle Fire, there's a layout for that. iPhone 3GS. It has a bunch of different presets for you to be able to play around with it.

But what I really want to show you here is if I enlarge it. Let's go ahead and set it back to the notebook. Close that. And I want to reload the page.

You notice it was tweaking the network there. So you saw the image kind of slowly load in.

But what I want to do is I'm going to bump up the font size. Let me make sure that I'm clicking in there. There we go. Now we're bumping up the font size.

So now, as I increase the font size of the text because I'm using ems, now the layout has switched to being the mobile layout as opposed to having the text run off the screen and having to move side to side. And you can see that the menu even popped in there. So that's one of the really beautiful things about using ems for your widths is that you can actually ensure that you're always providing the most appropriate layout given the font size and the amount of screen real estate you have available to you.

Now, general best practice is to minimize the amount of files that we're trying to include because every one of those files has to be requested and downloaded. So one way you can do that is to reduce your styles to simply two files or potentially even one, depending on how you want to go. But if you've got a lot of styles that you're trying to load in, two files might be just right.

So in this case, you see that we've got two stylesheets that are linked. We have our basic stylesheet and our advanced stylesheet.

```
…
<head>
  <meta charset="utf-8"/>
  <title>Page Title</title>
  <link rel="stylesheet" href="/c/basic.css">
  <link rel="stylesheet" href="/c/advanced.css"
        media="only screen and (min-width:20em)">
</head>
…
```

Browsers that don't understand media queries get basic.css, browsers that do get both.

And browsers that don't understand media queries, they see that only in the green stylesheets media query. They then ignore that stylesheet. So they would only get the basic.css file. But browsers that understand media queries that know what to do with only will go ahead and use that second stylesheet and download it in addition to basic.css.

So that's a quick and dirty overview of media queries. In the next chapter, we're actually going to dive into both responsive and adaptive layouts—to kind of describe how those work to you and show you how you can achieve those layouts very easily using media queries.

We'll see you in a minute.

RESPONSIVE AND ADAPTIVE LAYOUTS

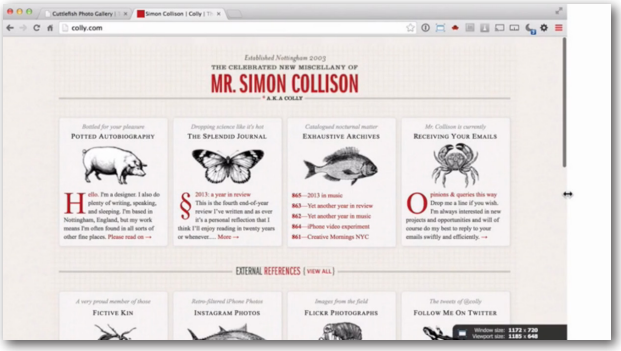
Welcome back. In this chapter, we're to talk a bit about responsive and adaptive layouts.

Now responsive web design was a term that was coined by Ethan Marcotte. And as he defined it, there are three things that make something responsive web design. And those are a fluid grid, flexible media, and use of media queries.

Now what do I mean by each of those? Well, for a fluid grid, what it means is that the grid that you're using to visually layout the design is percentage-based. So that means it's based on the width of the screen. And it's always proportional.

Now flexible media means that we don't have images and such that overflow their parent. So you're saying something like max width of 100% and a height of auto on all images. So jumping over to the browser here, if we inspect this, we've got all of these images laying out in their various columns. And if I shrink the document, you'll see I'm adjusting the layout to two columns, but the images never overflow, and then down to one column. And I'm using media queries to adjust how they're laying out, how they're floating next to one another.

So this is responsive web design. Now I'm using media queries there, but just using media queries is not responsive web design in and of itself. Let's compare this again. I'm going to shrink it down. You'll see that it's constantly filling up the full width. You can think of the browser as a container, and the contents like water being poured into it up until a sensible maximum.



Now here is Simon Collison's site. And as I shrink this down, it has three fixed widths. So the contents don't expand to fill. They simply go to three different sizes That is an adaptive layout using media queries, but it's not responsive web design. It's an important distinction.

Now the best practice is to use responsive web design from a mobile-first approach because it's the most economical. I showed you the example earlier of using the linked style sheets to deliver a baseline experience to every browser, especially older mobile devices that maybe don't understand media queries, and then using a media query to load in the additional content to layout the page as more screen real estate is available.

So that's a brief overview of responsive and adaptive layouts. In the next chapter, we're actually going to dive into some different stuff and look at using CSS transforms to adjust how elements are displayed. But before we jump over there, I'd like you to do Assignment 13: Design Alternate Layouts Using Media Queries.

So just like I talked about Stephen Hay recommending, I want you to shrink your browser width to roughly 320 pixels, which is about the size of most small smartphones, and then adjust your layout to provide a good reading experience. Then I want you to increase the width of your browser until the design starts to look not so great, and then use a media query to adjust the layout. And then rinse and repeat that pattern until you have a couple of different layouts for your design.

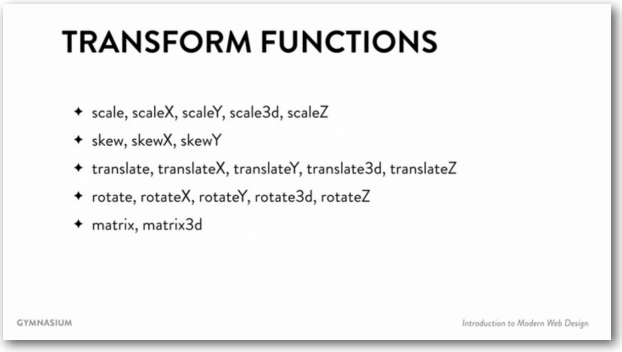
How many major and minor breakpoints did you end up with? What did you find challenging about this process? I want you to publish your work on GitHub and post a link to it in the forum, along with your observations and answers to these questions.

I'll see you in the next chapter.

TRANSFORMS

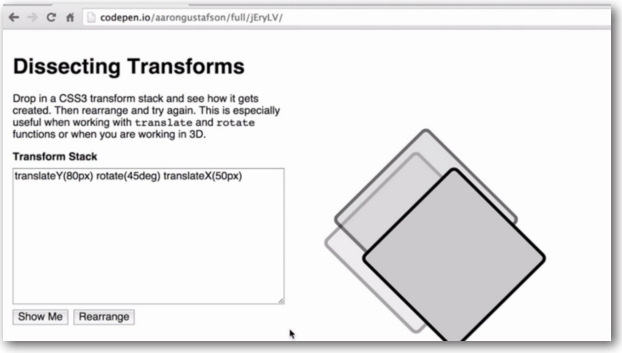
Welcome back to our discussion of CSS as a design tool. In this chapter, we're going to be talking about CSS transform.

The transform property takes a collection of transform functions as its value. And those functions allow you to manipulate the element in two dimensions or three dimensions. Some of those functions include things like scaling for enlarging and shrinking the element; skew, which allows you to distort the element by skewing it in one direction or another; translate, which allows you to move the element in any one of the x, y, or z axes; rotate, which allows you to, obviously, rotate the element in x, y, or z axis. And then it's got this special one called matrix, or matrix3d depending on which way you're going, that allows you to actually do matrix math for the transformation of the element.



Now the transform functions—it's important to note—are applied in the order that you write them. And so you can't use a transform and then unhover, have a transform, and add another transform function. You would actually have to write out each of the transform functions that you want. They're not automatically added together.

Let's take a look at a tool I put together for you that allows us to dissect how transforms are affected by the order of the transform functions that we choose to use. So here we see that the transform stack—so this is the value we would be using for transform—is translate x 50 pixels, rotate 45 degrees, and translate x 80 pixels. And the element ends up there.



If we click rearrange, it rearranges those three values. So now the rotation is first, then translate y, then translate x. When it runs this time, the element ends up in a different position.

If we are rearrange it a third time and run it, a completely different position. So this is a good demonstration of why the order of your transform functions really matters, especially when it comes to rotation and translation.

Now in the real world, the different transform functions do not animate in this way. I actually used JavaScript to add them one at a time in order to build up to where the element would be placed. But if you use a transform function, the element would just appear in the position that it would based on all of those transform functions being applied to it.

When it comes to transforms, especially in three dimensions, there are three other properties that often come up—perspective, perspective-origin, and backface-visibility. These are used a lot in things like card flips and other three-dimensional rotations of elements. We'll see some examples of these in the final chapter of this lesson when we dissect the Wikihow card game.

So in this chapter, we've covered transforms and all of the different transform functions. This was just an overview of what's available to us. We'll go into a little bit more detail in the examples at the end of this lesson. But in the next chapter, we're going to talk about transitions, which is how I got those nice little animations between the different transforms in that example. I'll see you in a few minutes.

TRANSITIONS

Welcome back. So transitions are one of these neat things that actually allow you to animate property values as they move from one value to another. And we achieve this using the transition property, and we set things like the property that we want to transition, what the duration of that transition should be, what function we want to use for that transition such as linear or ease in, out, et cetera. And then if we want any sort of delay before the transition should kick off, we can include that option as well.

So the most common use case for this is probably hovering links to give a little gentle movement between two colors as you're mousing over a link. So let's look over at the browser. Now, I'm using transitions in a few different places—here in the navigation, but for basically all links on the page.

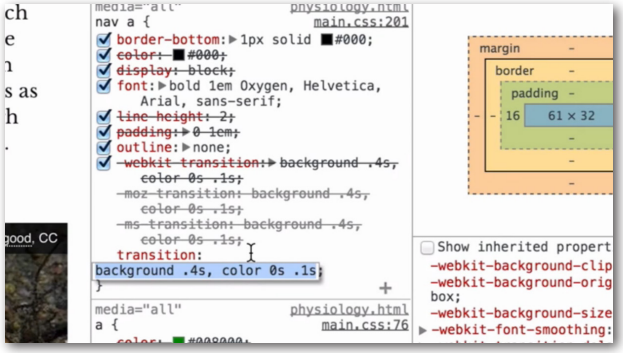
```
a {
  transition:  color      /* Property */
               2s         /* Duration */
               linear     /* Function (optional) */
               0.5s;      /* Delay (optional) */
}
```

Links are a common use case for transitions as it makes the typical :hover color change more subtle.

So if we inspect these, you see that the baseline color is white. That's when they're just sitting against the black. If I turn on the hover version, in the inspector here, you can see that the background changed to gray and that the color changed to black on hover.

Now, if we scroll down, we see for browsers that understand transitions—because they're a CSS3 property—will transition the background over 0.04 seconds. Now, why am I not translating the color and the background together? It looked a little weird in the navigation on a large screen.

So I'm only doing it on the small-screen version. That way, the color translation looks a little bit better against that big, thick, black background. Transitioning from the white to the black looked weird.



Let's scroll down the page here a little bit. And down here at the bottom, we've got the Wikipedia article on cuttlefish. And if I mouse over that, you see that it turns from green to red. If we look at that one, you see that I am transitioning to—or that the color is green. And then on hover, the color is red.

And I'm actually changing the border style too. It's transitioning both the background and the color. I'm not transitioning the border style, but I am changing the border style. So it goes from being a dotted border, which is very faint, to a solid border. So it gives it just a little bit more pop.

Now, the transition property is actually shorthand for four other properties—the property that you want to transition, the duration over which you want to transition it, the timing function, and then whether there's any delay.

And interestingly, as you saw in those examples, you can establish multiple transition effects at once using commas. This can be done in the individual properties themselves or in the roll-up of transition shorthand. So here, color over two seconds using linear, and then font size changing over five seconds easing in and out, and it's got a half-a-second delay.

So that covers transitions. And then in the final piece of this triumvirate here, we're going to talk about animations. So join me in a moment, and we're going to look at animations and how all of this stuff starts to roll up together. See you in a few.

ANIMATIONS

All right. Welcome to the last chapter in CSS as a Design Tool, the chapter on animations. This one should be a fun one. I know this has been a lot of information to take in.

So we're going to start out talking about @keyframes rule. @keyframes allows you to define animations.

And you do that by saying ThisIsMyAnimation (or whatever your name is that you want to give to this animation) after the @keyframes. And then inside of that, you have percentages. And those percentages are basically blocks that you can adjust different properties on a given element through the animation.

So here at the beginning of the animation, the top offset is 0 pixels. At 33% it's 50 pixels. And the animation-timing-function is being adjusted to ease-out. And then at 100%, the top is 100 pixels from the top and the left is 10 pixels.

```
@keyframes thisIsMyAnimation {
  0% {
    top: 0px;
  }
  33% {
    top: 50px;
    animation-timing-function: ease-out
  }
  100% {
    top: 100px; left: 10px;
  }
}
```

You can use percentage blocks to define what should be happening to the element at each stage.

You can use whatever percentages you want as you go around. But this is generally how you set up the keyframe animation. You can even redefine the animation within the animation, which is a little crazy.

So in order to use this animation once I've defined it using @keyframe is to go in and actually say, for this paragraph, I want to animate with this ThisIsMyAnimation. I want the duration of the animation to be four seconds. The timing function to be linear.

```
p {
  animation:   grow 4s linear,
               shimmy 2s ease-in-out;
}
```

And you can apply multiple animations at once.

GYMNASIUM *Introduction to Modern Web Design*

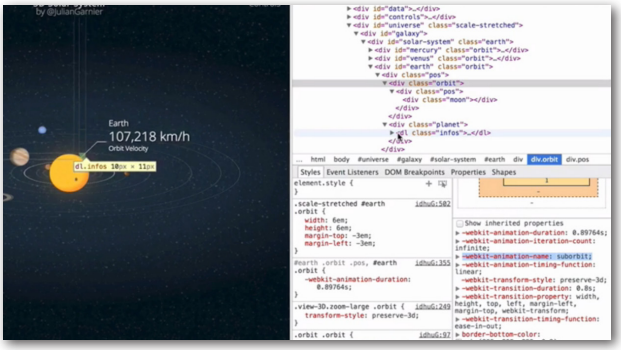
If I want a delay, I can say it. So in this case, 1.5 seconds. How many times do I want it to iterate? I want it to go infinitely in this instance.

Do I want it to reverse? Alternate would make it reverse. And then do I want it to fill any spaces in the animation? Basically as it's moving from the end of the animation back to the beginning again. And in this case, I'm saying "no." I've gone ahead and taken care of that.

And just like a lot of the other stuff we've been looking at in this lesson, you can define multiple animations for an element. So let's look at some crazy examples of animation out there on the web.

So this first one I'm going to show you is on CodePen. And in this case, we have Julian Garnier who has put together an entire operational solar system using animations. And if we tree down into Earth here and then we inspect that on the right-hand side, you can see web animation name is Orbit. In this case, animation is one of those that are still vendor prefixed as of the time of me recording it.

We have another on the actual position of the Earth. We have another animation called Invert. And then we have a Suborbit for the moon around the Earth. So you've got three different animations that are being used in here.



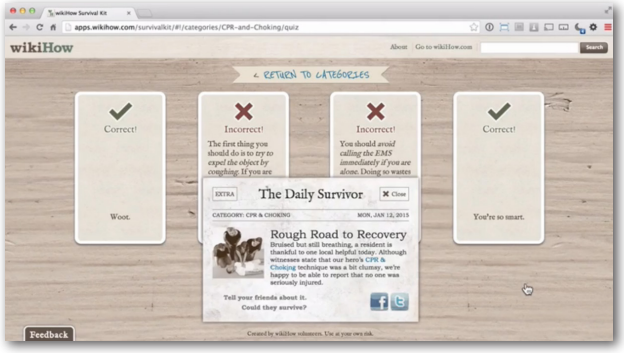
And then there are various other things that are being positioned and dragged around. And this is a really good one to kind of go into and look at all of the different stuff that's going on. But I specifically want to show you the different keyframes definitions.

So let's go over here. And we'll go into the page. All right. Here we go. Scroll down.

All right. There we go. Here are our different animations.

So keyframe for Orbit does a rotation in three dimensions from zero degrees to negative 360. Suborbit does a rotation of x and z. OK. And then Invert does an inversion of the rotation of x and rotation of y. So that's pretty interesting.

We could go in there and tweak that and adjust how each of those elements rotates. If you want to do that, feel free. The URL is up there on the screen.



The second example I want to show you comes from WikiHow. Now, this was a game that I was lucky enough to get to work on where we played with a lot of these animation and transition effects.

So you see things like card flip, which is where that backface-visibility comes in. And then things like laying out cards, shuffling them around, pulling up overlays. These all have to do with things like z-index and animating different properties.

Some fixed positioning going on there. Making different cards pop up, disappear.

I'll try the quiz on CPR here because this has some neat stuff going on, too. Here it's actually laying out the cards and flipping them over, enlarging them. We've got some drop shadows going on. All kinds of fun stuff taking place.

And I am not very good at taking quizzes on CPR. So I'm going to probably screw up a couple of these. And done. So rough road to recovery.

But this is a great example of triggering a bunch of animations and transitions, and transforms to create a more realistic, or, in some cases, referred to as skeuomorphic design. It makes it a little bit more lifelike. And this is a fun one to dissect.

And one of the things that we do a lot of in here to trigger the different animations and transitions and such, is swap class names on elements, which is something that we'll go into in the JavaScript lesson, which will be the next lesson that you'll be covering. We'll see how to swap classes on elements in order to do neat stuff like this.

So Jamie Hankins created an Easter egg in the DNS entry for jamiehankins.co.uk so that when you loaded it up in something like who.is, you would get a little bit of "Harlem Shake."

[MUSIC - BAAUER, "HARLEM SHAKE"]

So that shows just how silly you can be using CSS animations and music—even in putting Easter eggs in your site.

So in this chapter, we covered the @keyframes rule and the animation property. And what I'd like you to do for your final assignment in this lesson is to look for ways to include transforms, transitions, and animations in your site. Take a look. See what happens when you transition certain properties like width for every element on the page as you resize the browser. That can get kind of interesting.

Look for inspiration on the web. Maybe pick apart some of the examples I've shown you. And post your findings and some of the stuff that you've done in the forum.

And then last but not least, I want you to take Quiz Number 4, which will go through and assess your retention of all the things we've talked about here in Lesson 5.

I'll catch you again in Lesson 6, where we'll start to talk about JavaScript.