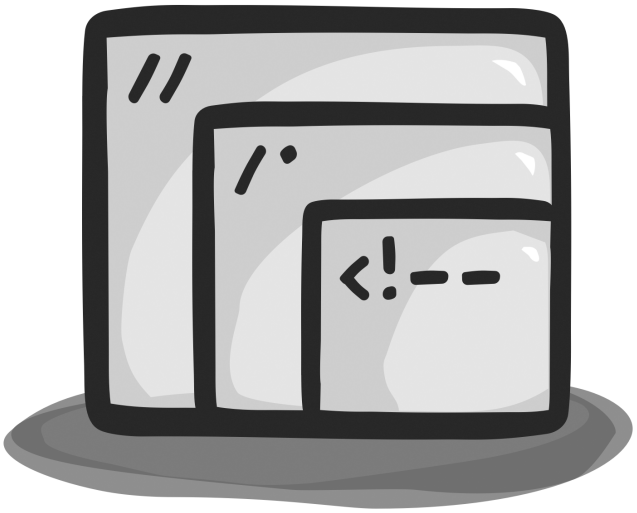


GYMNASIUM

INTRODUCTION TO MODERN WEB DESIGN

Lesson 2 Handout

An Introduction to HTML

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. When we talk about HTML, we often use the term markup. Markup is tagged content. HTML elements are made up of tags which typically (although not always) include an opening tag such as and a closing tag such as .
2. HTML elements sometimes have attributes. Attributes contain information about the particular element. Here is an example of the image element with a src attribute: <img src="gouda.jpg">. Attributes are only applied to the opening tag. You will never see an attribute applied to the closing tag of an element.
3. In HTML, elements are nested within one another, and this create relationships between those elements. We often use family metaphors in order to discuss how elements are related to one another, such as "parent" and "child." The relationship between nested elements is known as the "Document Object Model" or (DOM).
4. Semantic HTML is the term used to describe documents that are constructed with meaningful and appropriate tags, the goal of which is to provide more context to people who are reading them. Additionally, HTML has a limited number of elements available for use, and sometimes we need to extend the meaning of a particular element; in order to do this, we can define a "class" attribute—for example .
5. URL stands for "Uniform Resource Locator", commonly referred to as the "hyperlink." Different types of URLs include relative URLs, absolute URLs, and fragment identifiers.
6. The most common way to embed images in HTML is the use of the <img> tag. In recent years we have also seen the addition of the "src" and "srcset" attributes for the <img> tag, and these give us more options for providing the best image for the job when it comes to mobile devices and other contexts.
7. Adding media to your web page begins with the relatively new <video> and <audio> elements. These elements allow you to embed video files such as the MP4 format and audio files such as the MP3 format. However, you will almost certainly need to provide additional fallback content to account for different browsers and/or older browsers.

ASSIGNMENTS

1. Quiz
2. Begin marking up the content from the article you chose on Wikipedia in Lesson 1. The goal is to create a bare bones HTML document that will become the foundation of your website. Here are the specific steps:

    - Create a blank text document with an HTML extension, paste the text from the Wikipedia article you chose into that document, and open it in a browser, and see how it looks.
    - Return to the document and begin adding different elements like <html>, <head>, <body>, and <title>. Save it, view that page in the browser and see how it's changed.
    - Go back and add some text-level semantics—for example <emphasis> or <i> or <b> or some links. Continue refining the document and keep looking at the browser and see how it's changed.
    - When you're happy with your file, go ahead and push that up to Github and post a link in the forum. Your Github repository should look a little bit something like this, where you have an index.html file in it.
    - The HTML file will probably end up looking something along the lines of this. I look forward to seeing you in the next lesson, where we'll dive a little bit deeper into HTML5 block level elements and some of the other options available to us.

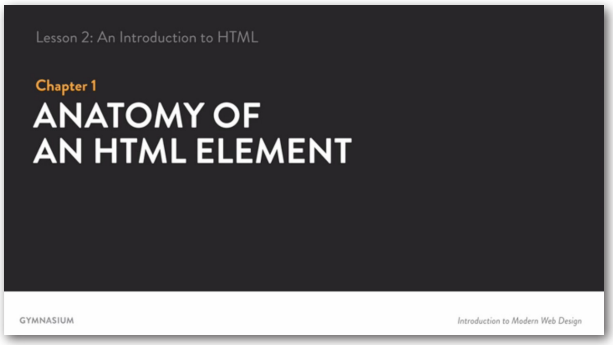
RESOURCES

- File Format plugin that opens and saves 24-bit WebP images
  http://telegraphics.com.au/sw/product/WebPFormat
- A presentation by Jenn Lukas on the various implications of dealing with images,
  http://slideshare.net/JennLukas/lets-talk-about-responsive-images-and-performance
- Browser compatibility for media formats
  https://developer.mozilla.org/en-US/docs/Web/HTML/Supported_media_formats#Browser_compatibility
- HTML elements belonging to the flow content category
  https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Content_categories#Flow_content
- Hyperlinks
  https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Hyperlink
- The HTML <img> element
  https://developer.mozilla.org/en-US/docs/Web/HTML/Element/img
- The HTML <video> element
  https://developer.mozilla.org/en-US/docs/Web/HTML/Element/video
- The HTML <audio> element
  https://developer.mozilla.org/en-US/docs/Web/HTML/Element/audio

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.)

Welcome back to Lesson 2. In this lesson, we're going to start talking about HTML, which is one of the foundational languages on the web. And in this first chapter, we're going to talk about the anatomy of an HTML element.



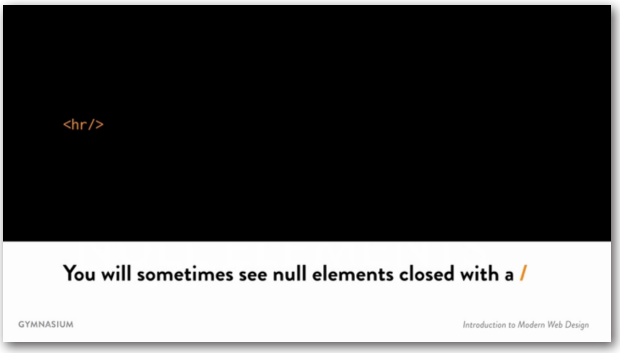
Now, when we talk about HTML, we often use the term markup. Markup is tagged content. HTML elements are made up of tags.)

So here we have an example of an h1 element. And the h1 element has two tags, an opening and a closing tag, that appear between the opening and closing angle brackets. (Or less than and greater than symbols.

Now, the first one we have here is the opening h1 tag. The second one is the closing h1 tag. And what happens is that the first tag tells the browser to start the element. And the second tag tells it to finish the element.

Elements are made of tags. Most elements that we see in HTML have two tags like this h1 does. Although you will find certain elements like this hr, which we'll talk about the meaning of a little bit later, have only one tag.

Elements that only have one tag and have no content are considered null elements. You might sometimes, when you're browsing other people's websites and viewing the source of their HTML, you may see some of those null elements with a trailing slash inside. This is a carryover from the days of XHTML, when we wanted to explicitly close every element. So we would say that this element is self-closing because it both starts and closes the element,



Now, HTML elements sometimes have attributes. Attributes contain information about the particular element. So here we have an image element.

And don't worry about fully understanding what this markup means right now, but here we have an image element which contains two attributes. An src attribute with a value inside the quotation marks after the equal sign. And then an alt attribute for the alternative text, which is inside of quotes as well.

Now, these two attributes provide information about the image. The src is the source file for the image that should be displayed. And the alt, as I had mentioned, is the alternative text that should be displayed if the image is not available.

Attributes are only applied to the opening tag. You will never see an attribute applied in the closing tag of an element.

So for instance, this h1 element has an ID of logo. An ID attribute equal to logo. And that ID attribute should be on the opening tag.

Now, some attributes have values while others don't. Here's another example of an HTML element. In this case, it's an email input type. And this one has three attributes.

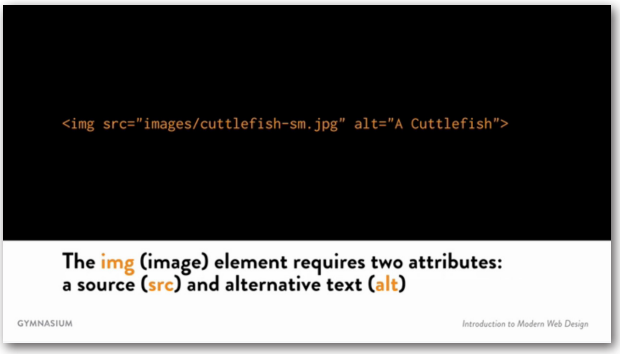
It has a type attribute saying that this is an email type field. It's got a name, which is the variable email_address, which will be passed to the back end, and then we have the required attribute, which indicates that this particular field is required in order for the form that it's inside of to be submitted.

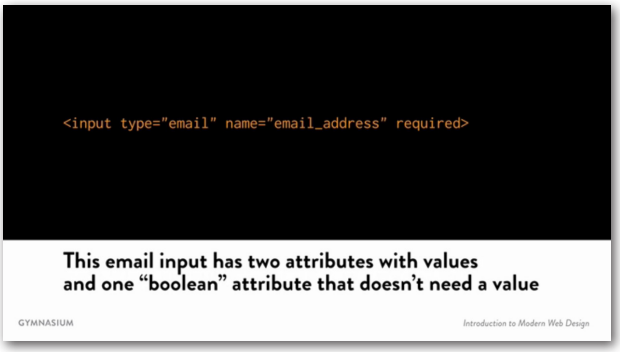
Now, the required attribute is an example of a boolean attribute. And a boolean attribute doesn't need to have a value. Here we don't have required set equal to anything. But you could just as easily say required=" ". Any of these is equivalent, but the simplest way to say it is simply required.

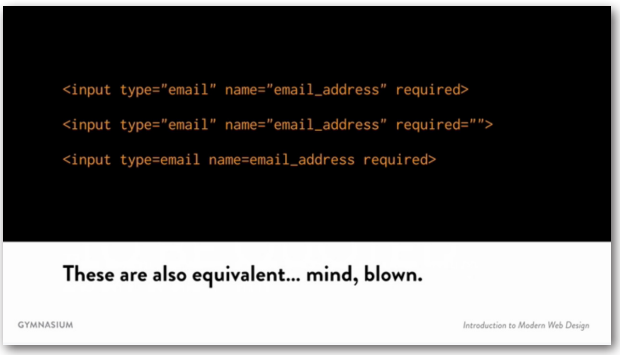
It's also worth noting that attribute values without spaces in them don't actually need to be quoted. So if we look at three different versions of that same input, all of these are equivalent.

In the first one we have our first two attributes with quotes around them, and then the required attribute shorted. The second one is the one we just saw were required=" ". And then in the third example, all three attributes have no quotes around the value of the attribute.

This is perfectly valid as long as the value of the attribute does not contain any spaces. Because if it was to contain a space, then the browser wouldn't know what the demarcation point between the different attributes would be. So the second word in that value would be treated as though it was a new attribute.







Now, throughout this course, I'm going to bring up a lot of best practices. And you'll see a slide like this when I want to introduce those to you. Now, the best practice that I want to impart to here is that it's important to keep your markup consistent.

So this, for example, is perfectly valid HTML. Here we have an h1 element with an ID attribute, with a value in it like we've seen with all the quotes and stuff. And it's all in lowercase.

Now, in the second thing, the jump to the menu (an element), you see the A is capitalized. The ID is capitalized. There's no quotes around nav-jump. And then there is single quotes around value #menu.

This is completely valid HTML, but it's not very consistent. The markup is all over the place. And so coming into this file and looking at it, it looks like three or four different people potentially marked up this document. There's no consistent voice to it. For that reason, we recommend that all of your HTML be marked up in the same consistent manner.

And the manner that we overall as a profession have settled on is to use all lowercase elements and attribute names. And to always quote the values, most often with double quotes.

So in this chapter, we've covered what tags are, what HTML elements are, what attributes are, and the importance of consistency in our markup. In the next chapter, we're going to talk about how elements can be nested within one another, what the Document Object Model (or DOM) is, get to know a little bit more about fault tolerance in HTML, and finally, talk about the semantic nature of HTML.

I'll see you in a few minutes.

HOW ELEMENTS COME TOGETHER

Welcome back to Chapter 2. And in this chapter, we're going to be talking about how elements come together. Now, in HTML, one element can be nested within another element. In fact, multiple elements can be nested inside of other elements.

In this particular example, you see an h1 element and an a element nested inside of a header element. So these two elements would be considered children of the header. They would also be considered siblings of one another. We often use family related metaphors when discussing HTML elements.

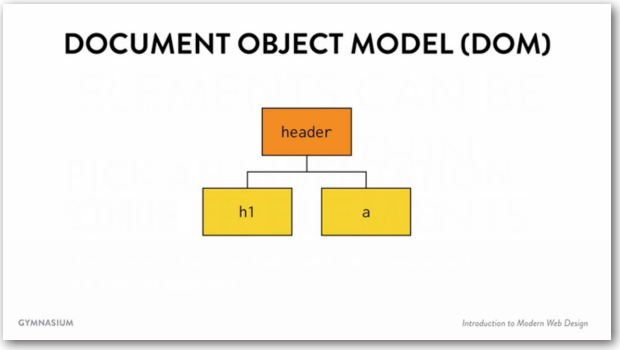
Now, when working with markup and nesting elements, it's important to keep things organized in order to best be able to read a document. In this case, you see I've indented the children within the parent.

In terms of organizational best practice, it's just a good idea to pick an indentation style and stick with it. Different organizations have different indentation styles. Some use spaces. Some use tabs.

There is no right way to indent except to indent. And to make sure that your team or the people that you're working with are consistent from document to document. Because again, coming in and being able to see a consistent indent style makes it easier to edit files later on.

So if you like using two spaces to indent, great. If you like using a tab to indent, perfect. Pick a style and go with it.

Now, when we nest elements within one another, we create relationships between those elements. We often use family metaphors in order to discuss how elements are related to another. And when we diagram how they're related, we often use a tree.



So here you can see the header element is the parent. And it has two children, the h1 and the anchor element. Taken together, these form the Document Object Model.

Now, what's particularly interesting here is that the header element was introduced in HTML5. So older browsers don't know what to do with the header element. But thankfully, there's a rule for how browsers handle elements that they don't understand. They ignore them.

Now, how this plays out in terms of our markup example is that browsers that understand the header element will see the entirety of this markup. They'll see a header. And then inside the header they'll see an h1 and an anchor. Browsers that don't understand the header element will ignore it entirely but they'll expose the contents inside. This is an important feature of HTML, and it's what allows the language to evolve over time.

Now, when choosing elements within HTML to mark up our documents, we should choose elements that make our content more meaningful. HTML is intended to make our documents more expressive. Here we have just a normal passage of content from my cuttlefish page. And if we were to do this in the browser, this is what it would look like. Straight, normal Times New Roman.

Now, if I wanted to start marking up the document I might decide to link a brown pigment to the Wikipedia article on Sepia. And when I do that, now all of a sudden that becomes a link within the document. That's what the anchor tag does. And we'll talk a little bit more about the anchor tag later on.

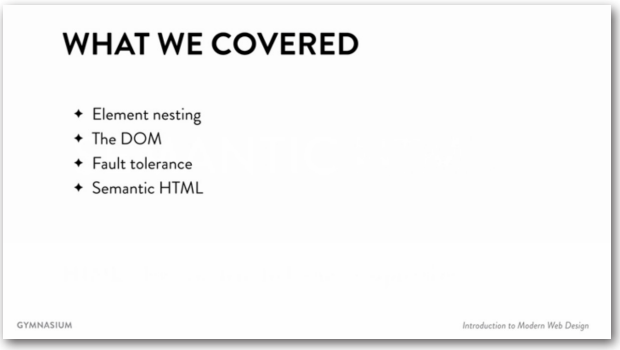
Continuing on, if I wanted to call out certain terms as being in an alternative voice, alternate mood, or an alternate language, I could use the i element. Now, as you see here, I've got several instances of the i element marked up in green. And each of them has a lang attribute, which indicates the language that the term or terms within the element is in.

Again, flipping over to the browser, all of these i elements are rendered in italics, which visually sets them apart from the normal text of the page. But the markup actually has additional information inside of it that provides context around the language that that particular term or terms are in.

Taking things a step further, this entire paragraph is a paragraph. Therefore, I can wrap it in a p element. And then that will insert some margins above and below this particular block of text.

All in all, HTML is intended to allow our documents to be more meaningful, to provide more context to people who are reading them. This is what we refer to as semantic HTML.

In this chapter, we talked about element nesting. How elements come together to form the DOM. We talked about fault tolerance, which will come up time and time again as we're discussing HTML and CSS. And then finally, we talked about the importance of semantic HTML to provide more meaning to our documents.

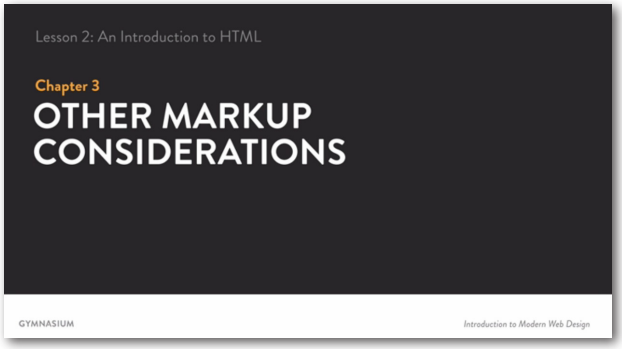


In the next chapter, we'll wrap up a little bit more of the introduction to HTML with talking about how to comment your HTML, and what's meant by character encoding.

I'll see you in a few minutes.

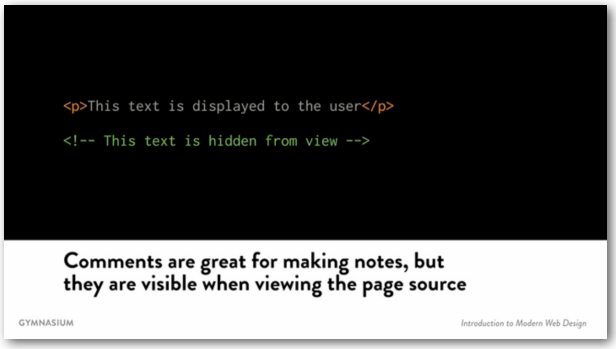
OTHER MARKUP CONSIDERATIONS

Welcome back. In this chapter, we're going to talk about a few more markup considerations just to kind of round that out before we start diving into the different elements that we have available to us in HTML. The first topic I want to cover is HTML comments.



With HTML, you can comment your code and leave little notes for yourself, or for other people that might be editing your document. In this markup example, we see a paragraph followed by some other code that's in green. Now, the code that's in green starts off with a less than symbol or an opening angle bracket, an exclamation point, and two

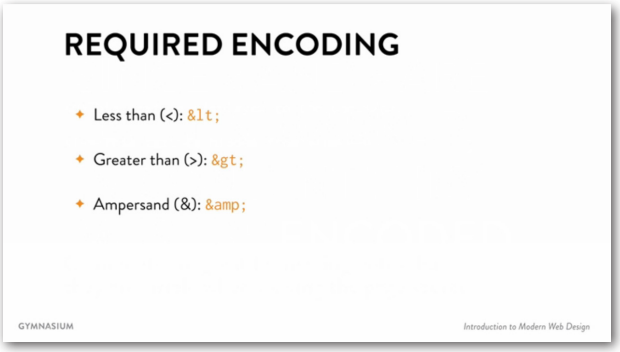
dashes or minus signs. It then contains text. This is text hidden from view. And then it finishes with two dashes and a greater than symbol, or a closing angle bracket.



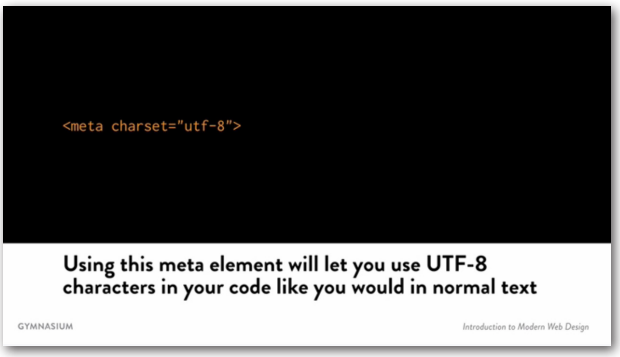
This is a comment. The text within the comment will not be shown in the browser. Comments can be really useful for debugging, because if you have an element in the page and you're not sure if it's causing an issue with the layout, you can always comment that element out temporarily and see if it fixes the problem. If it does, then you uncomment it and make whatever adjustments you need to.

Now, as we've been looking at all of this markup, we have seen the opening closing and angle brackets or the less than and greater than symbol frequently used with our tags and other markup-related bits. Since they're used so frequently in markup, these are characters that must be encoded, because if we just put them nakedly in our document, the browser might think we're trying to open an element.

Character encoding in HTML is accomplished using the ampersand and semicolon characters with text in between. So here you see the ampersand lt semicolon for the less than symbol, the ampersand gt semicolon for the greater than symbol, and the ampersand amp semicolon for the ampersand itself. Because ampersands are used to kick off an encoded character, we need to encode that character as well.



So here's a quick example. Everyone loves fonts by Hoefler & Co. In this case, because Hoefler & Co. contains an ampersand, we need to use the encoded ampersand character. Now, when it comes to authoring your documents, if you use the meta tag to indicate the character set that you're using, you can use any UTF-8 characters throughout the document, with the exception of the less than, greater than, and ampersand characters.

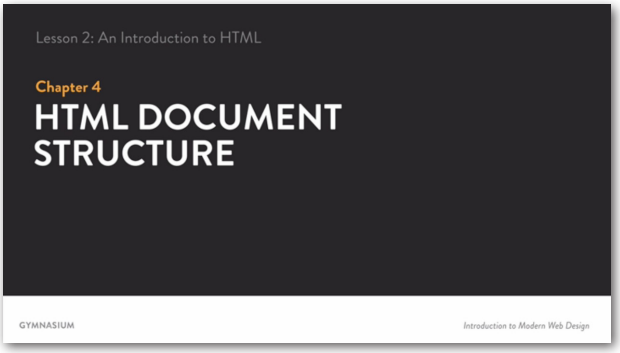


But if you want to be more specific, however, you can use encoding for a variety of characters. There's a complete table available at the ASCII site. And here's a brief overview of some of the characters that are available, and their HTML numeric and named entities. But as I said, if you're using the meta tag to indicate the character set that you're using for the document, all you really need to worry about is the less than, greater than, and ampersand characters.

So in this very brief chapter, we talked a little bit about commenting your HTML and character encoding. In the next chapter, we're actually going to dive into a lot more detail and talk about HTML document structure, the title element, and meta tags. I'll see you in a few minutes.

HTML DOCUMENT STRUCTURE

Welcome back to Chapter 4 of an introduction to HTML. In this chapter, we're going to talk about HTML document structure. Now, here, we have what is possibly the simplest HTML document possible. At the very top, we have the DOCTYPE and the DOCTYPE says that this is HTML.

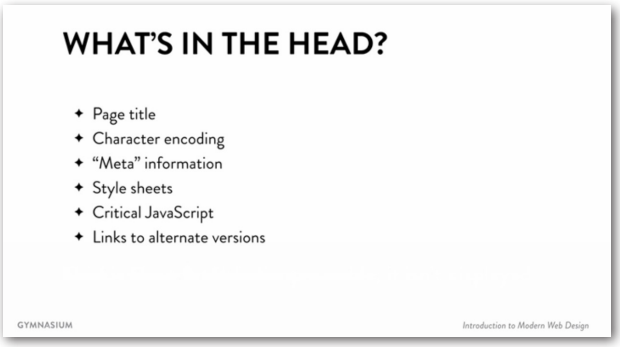


Now, there are a variety of DOCTYPEs available that specify the particular flavor of HTML or perhaps XHTML that's being used for the particular document. What you see here is the HTML5 DOCTYPE, which is perhaps the simplest DOCTYPE and the one you'll use most often. Now, the HTML element is the element that needs to come first in your document. It is also referred to as the "root element" because it's the first element on the page, and all of the other elements are descendants of it.

Now, within the HTML element, we can and should declare the language that the page is written in. So in this case, I am saying that the language of the page is English.

There are a variety of other language codes available as part of the International Standards Organization's 639-2, and here is a link to those. Now, within the HTML element, the first child that we have is the head element. You can think of the head as kind of a preamble or a foreword for the page.

It isn't displayed, but it contains all sorts of information about the page. Examples include the page title, the character encoding, and other meta information, style sheets, critical JavaScript, and links to alternate versions of the content. I'll walk through a couple just to give you an overview.

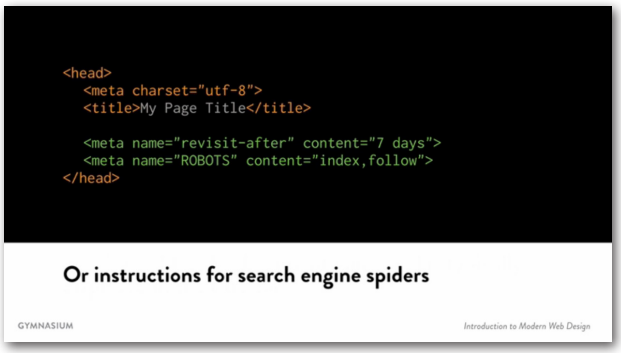


Title, for instance, is used for the document name, and this is what is displayed in the title bar. So here we see in the browser, it says "My Page Title."

Now, the head element is also where we would put that meta tag for defining the character set that we're using for the document. Typically, you want to put that before the title because that way you can use UTF-8 characters within the title as well.

There are a wide variety of other meta elements available to you as well, and these meta elements are used to provide supplementary information—or meta information—about the document. One example would be the page author or perhaps some instructions for search engine spiders. Don't worry about not understanding these. I'm just giving you a couple of examples.

Another one would be writing instructions for mobile devices to tell them how to handle the page, or you might use a collection of meta elements to supply information for social sharing and the like. Basically, any information about the content should go into the head of the document.



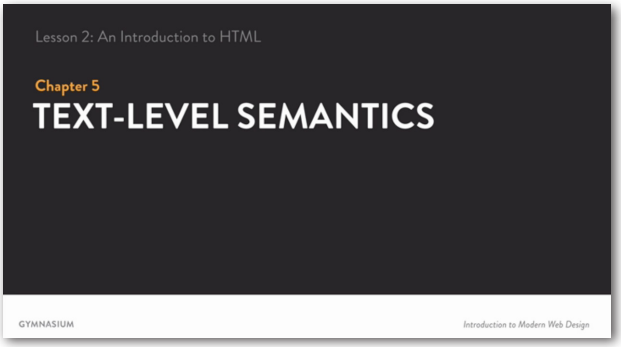
Another example of this is using the link element to indicate alternate versions or resources for the document. In this example, I'm using a link element to point to the canonical reference for my website on cuttlefish. Since the page content comes from the Wikipedia article on cuttlefish, I am referencing it within a link element and saying that the relationship of that link to this page is that is the canonical reference for this page.

Now, after the head element, we have the body element. This comes second within the HTML element, and the body is where all of your displayed content goes. Now, in terms of marking up a page, this is all you really need. But as you'll see in the coming chapters, this is really just the beginning.

So in this chapter, we covered HTML document structure, the title element, meta tags, and we also threw in a link element in there as well. In the next chapter, we'll start actually marking up some content and talking about text-level semantics—or phrasing elements. I'll see you in a few.

TEXT-LEVEL SYMANTICS

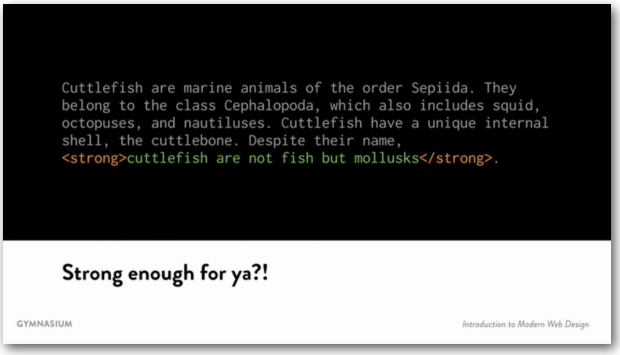
Welcome back. In this chapter, we're going to start actually looking at markup and how to markup text. We'll be talking specifically about text-level or phrasing semantics.



So let's look at an example of content. Here we have a simple paragraph of information about cuttlefish. Couple of sentences, some commas, some conjunctions. All that sort of good stuff. Now, looking at this, this little bit at the end here—"cuttlefish are not fish but mollusks"—seems kind of important.

Now, HTML supports a couple of different ways of increasing the importance of a given phrase or word. The first I want to talk about is the strong element, which indicates that certain content is of strong importance. So if we were to wrap a strong element with an opening and closing strong tag around “cuttlefish are not fish but mollusks,” we would be saying that this is something that is of strong importance.

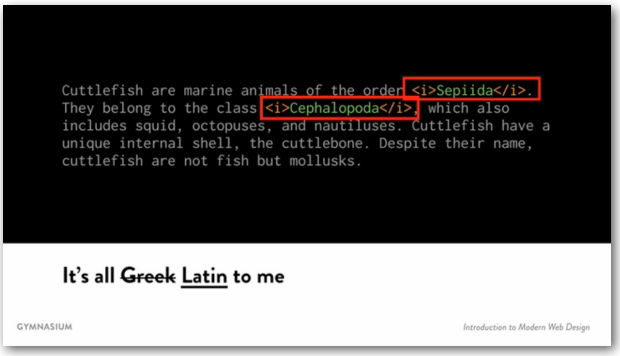
If you think about this being read aloud, you might think of something like “despite their name, cuttlefish are not fish but mollusks.” Now, the default display of this in a browser is to make it bold. Using CSS we could make it look any other way we want to, but that’s the default appearance in a browser.



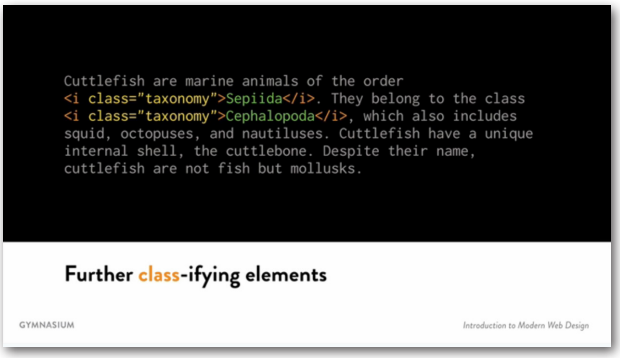
Now another way that we could emphasize a particular phrase is by using the emphasis element. The emphasis element is a little bit different than strong importance because it’s just trying to emphasize in some way. So “despite their name, cuttlefish are not fish but mollusks.” So it’s a slightly different inflection; we’re emphasizing it but it’s not really different in terms of being of strong importance. It’s just something that’s being emphasized. There’s a little bit different intention behind the use of emphasis versus strong.

Now in terms of the display in the browser, here you see that the emphasis element gets displayed in italics by default. Now the I element, as we’ve discussed earlier, is used for text that’s in an alternate voice or mood, or even an alternate language.

So we saw before, with some of the other examples of words that were in another language, were highlighted using the I element. So here we have “Sepiida” and “Cephalopoda” and these are Latin terms, so we might want to mark those up in I elements. And then those also become italicized automatically. And as I mentioned in the last chapter, we can use the lang attribute to indicate that those are actually Latin.



Now the class attribute is used to extend to the meaning of any element. So in HTML, we only have a limited number of elements available to us, and sometimes we need to extend the meaning of a particular element in order to use it in a specific context. So I’ll give you an example. We had our “Sepiida” and “Cephalopoda” terms, which were marked up in I elements. We could classify those as being taxonomic terms. So not only are these potentially in Latin, which is not displayed here using the lang attribute, but we could say that these terms are being used specifically having to do with taxonomy.



And then, if we wanted to, we could further classify those elements as being the order and class specifically. So taxonomy, order, class—these are not elements that exist in HTML, but we could indicate using classification that these I elements are for taxonomy. Together, each one is for taxonomy—they both share that—but then one is specifically classified as an order, and one is specifically classified as a class.

We can use these classifications for styling purposes, for scripting purposes, or in order to expose information from the page to other technologies. In terms of the actual display, they still look like italics, but we've added a little bit more semantic meaning to the document by indicating how these particular elements are operating.

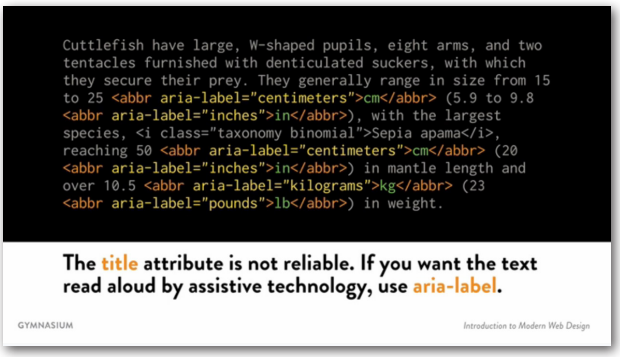
Now in terms of best practices, it's important to keep your class names meaningful. You don't want to classify something as big red text because down the road, maybe the design changes and now all of a sudden that is small blue text. So if we use class names that are meaningful and actually have some semantic value to them, then they will be able to persist, even if the actual display, the style of that particular element changes over time.

So let's look at a new passage here. Here we have another passage from the cuttlefish website, and we see a familiar friend in there, an I element with a class of taxonomy, so it shares that with the other two instances of I that we saw earlier. But in this case, we're dealing with a binomial, which is the genus and the species, Sepia apama.

But I don't want to focus on that, I want to look at a different challenge. So here we have a couple of different phrases that have to deal with measurements. And specifically, we have these abbreviations for elements cm, in, kg, lb. And when we extract those out, really what those mean is centimeters, inches, kilograms, and pounds.

Now in HTML, we actually have an element that allows us to expose this information to users. And that's the abbreviation element, or abbr. It indicates abbreviations and then gives an expansion for that particular abbreviation. So in terms of these four abbreviations, what we're looking at is an abbreviation element with a title that gives the expansion of the term. So abbreviation title equals centimeters for cm, abbreviation title inches for in, and so on and so forth. If I drop them in context, it would look something like this.

Now hopping over into the browser, what you see is that this title information actually gets exposed as tool tips. That's pretty awesome. Now unfortunately, from an assisted technology standpoint, that title information is not read out to users, at least not reliably. Most screen readers ignore title attributes unless specifically instructed to read those out.



In terms of assistive technology, if we want it to actually read out 15 to 25 centimeters, or 5.9 to 9.8 inches, then we need to go an alternate route. And this is where aria-label comes in. Aria-label is actually read out as the label for this particular

element. So when the element's encountered, cm, the text "centimeters" would be read out instead. Let's look at a quick example.

"They generally range in size from 15 to 25 centimeters." Perfect. That's exactly what we wanted it to sound like. Let's say we are dealing with a reference to a particular publication, such as the *Journal of Experimental Biology*, as you see here. HTML has an element for referencing publication names, titles of films, and the like. And that's the cite element.

We can wrap the cite element around the *Journal of Experimental Biology*, and then we have indicated that this is a citation. We could take it a step further and perhaps classify this particular citation as being a journal name. Or you could say a magazine, or if it's a book title or a movie title, maybe you give it a classification based on that, in case you want to style each one differently.

In terms of visible display, cite automatically is given italics in pretty much every browser. Now sometimes, when you're creating an HTML document, you might be discussing something like, well, HTML, as I'm doing here. Now here's an example of that. I'm a big fan of the title attribute, but I wish it was better supported by screen readers. So here I'm discussing code, and this is where I could actually use the code element to indicate that this is in fact code.

Again, I might want to take it a step further and classify this code that I'm talking about as being HTML. This is just another example of how classification can further extend the semantics that are built into HTML, to indicate the sort of content that's being contained inside the element. The code element will be displayed in a monotype font, usually a slab serif, so that it looks like text from a typewriter.

One of the new HTML5 elements actually allows us to indicate times, dates, and the like on our pages. So here we see 15 August 2014. As you may have guessed, this is the time element. Now the time element can simply be wrapped around the date, or you can take it a step further with the date-time attribute and actually provide a computer readable date in the International Date format of year, month, day separated by hyphens.

Now I've talked about a bunch of different options with respect to HTML that exists—so tags that are available to us, elements that exist within HTML—but sometimes those semantics aren't enough, even with classification, to do what we want them to do. So how do we deal with arbitrary bits of text?

Here we have "An Introduction to Cuttlefish." If I wanted to select a portion of this, I could use a span element. This is used for arbitrary spans of text. So if I wanted to only show the word "introduction," as opposed to "An Introduction to Cuttlefish." I could wrap the stuff that I don't want to show in a span element. So here we see a span element around "an" and the space, and a second span element around the space and "to cuttlefish."

I've included the spaces inside of those elements as opposed to around "introduction," in order to ensure that those spaces are not exposed visibly in the browser. And the reason is that I'm going to classify these spans as hidden, and then use CSS to make them disappear. If I left the spaces outside of those elements, there would be a space both before and after the word "introduction," which may not be desirable in terms of the actual display of the page.

Now I'll show you an example of how this plays out momentarily, but let's look at another option for arbitrary spans of text. The B element used to be for bold, but in HTML5 it was actually switched to mean that it's text that's offset in some way, but is of no greater importance than the text surrounding it. In other words, they made the B element essentially a shorter span.

So I could just as easily say B class equals hidden, instead of span class equals hidden. Now I've used this approach on the fascinating cuttlefish site in the navigation. Let's look at that example. So here you see, if I inspect the introduction navigation item and expand that link, you can see it actually says "An Introduction to Cuttlefish," even though visibly, all we see is the word "introduction." that's pretty cool.

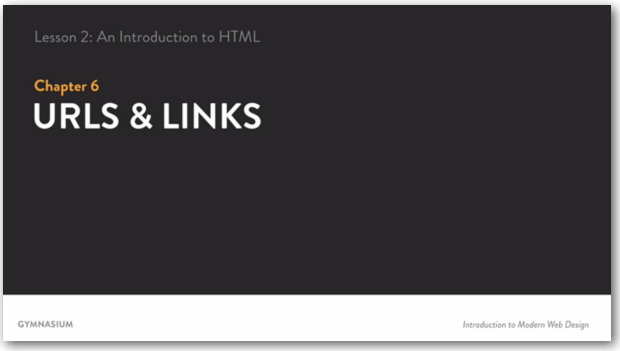
The B element can also be useful when we want to highlight the name of a person, or a product, or something like that. Again, text that's set off in some way as being different than the text surrounding it, but it is of no greater importance. So here we have Nick Hobgood and he is a person. So I can give him a class of person and use the B element there.

So in this chapter, we've talked about a variety of text-level semantic elements, such as strong, emphasis, I for an alternate voice or mood or an alternate language, how to use classification to extend semantics, abbreviation elements, the cite element, code, time, and then arbitrary spans of text using span and B.

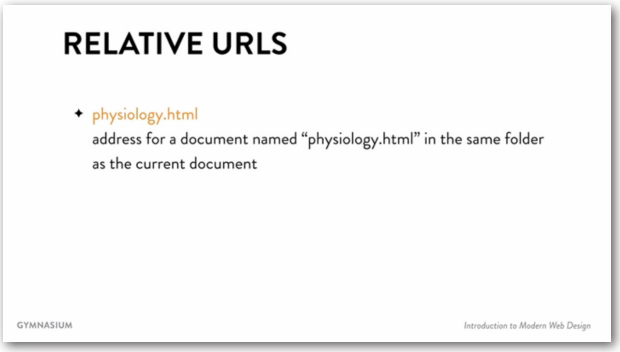
In the next section, I'm going to talk a little bit about URLs, links, and file references, before we dive into more organizational semantics.

URLS & LINKS

Welcome back. In this chapter, we're going to talk about URLs and links. So first of all, what is a URL? A URL is a Uniform Resource Locator. In other words, it's an address on the web. Sometimes you'll hear them referred to as Uniform Resource Identifiers, but most often you'll hear URL for Uniform Resource Locator. In other words, it's where documents live.



Now, there are many types of URLs. We have relative URLs, absolute URLs, and then we have fragment identifiers. I'm going to walk through each of these different types just so that you can fully understand how they work. First of all, we have relative URLs. The most common of these, you'll see one document pointing to another one.

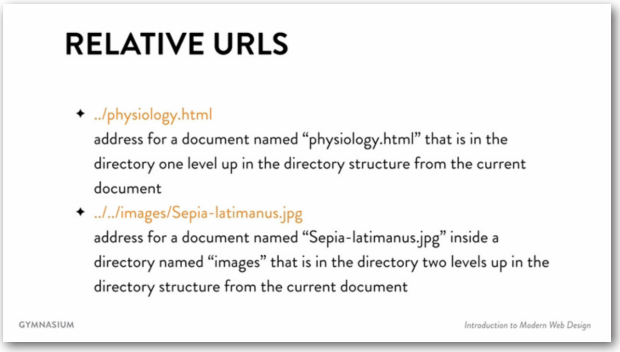


In this case, this URL is pointing to a document named physiology.html. And most servers are case sensitive, so we need to make sure that the document name is actually physiology.html, all lower case. And it's looking for that file in the same directory as the file we are currently in. So if we were to look at a file system—let's say we were in the index.html page, and we're referencing physiology.html—it would be in the same folder.

Now, here's another example of a relative URL. We have images/Sepia-latimanus.jpg. Now in this case, this is an address for a document named Sepia-latimanus.jpg inside of a directory named images that's in the same directory as the current document. So again, looking at the file system, if we're on the index page, we're moving into the images folder, and then to Sepia-latimanus. And notice that the case of the link, the capital S, actually matches the capital S in the URL.

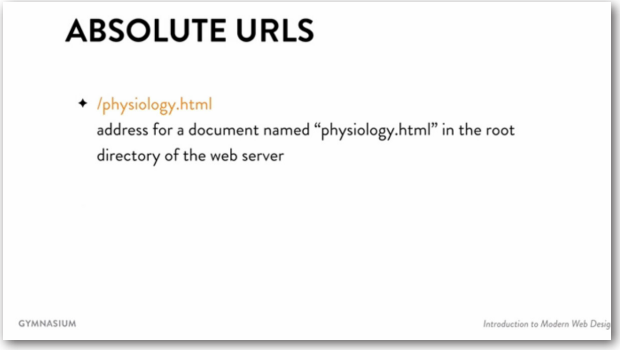
Now, another kind of relative URL that we have used is the dot forward slash. You don't see this all that often, but dot forward slash simply means start where this document is and then move. So in this case, ./physiology.html is the same as simply saying physiology.html. ./images/Sepia-latimanus.jpg is the same as simply saying images/Sepia-latimanus.jpg.

Where things get a little bit different is when you've got dot dot slash, as you see here. What dot dot slash means is, start where you currently are, move up one level in the directory structure, and then look for the document. So in this case, the file physiology.html should exist one directory level above the document that's referencing it.

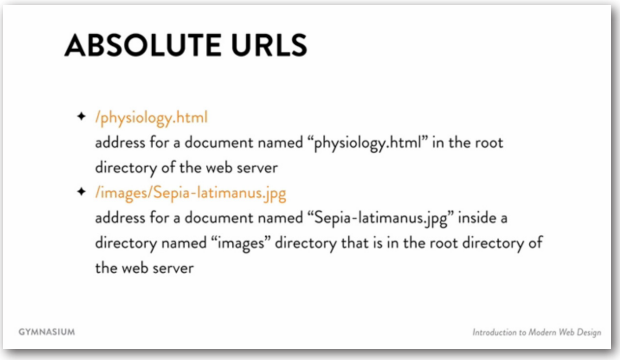


So in terms of the file system that we've got here, you might have a file inside of videos, and it's referencing physiology. So in this case, it needs to move up a level because it's in a subfolder. Taking it a step further, here we have ../../images/Sepia-latimanus.jpg.

In this case, we're moving up two levels in the directory structure, and then moving down a level into the images folder, and then to the file. So in terms of our file system, we might be inside of a folder within the videos folder, so we want to move back up to the videos folder. And then we want to move out to the outer folder, and then into the images folder, and then to the actual image that we're looking for within that folder.



Relative URLs, when you start using dot dot slash, and so on and so forth, can get a little confusing, which is why we have absolute URLs. Absolute URLs begin with a forward slash. Now, the forward slash simply indicates that the URL that you're referencing starts at the root level. And that means the primary folder that your site exists within. So if I say /physiology.html, it doesn't matter where the file is that's referencing this URL in this way exists in terms of the directory structure; it simply points to a physiology.html file that exists in the root folder of the site.



Now, if we were to reference /images/Sepia-latimanus.jpg, this means that the image file that we're referencing would exist within an images folder, which is in the root of the document. So it doesn't matter if we're two levels, five levels, 18 levels deep in the directory structure of the website; the image file is always within the images folder in the directory root, just like that.

So absolute URLs are a little bit better because they're more predictable. And it means that files are more portable within the overall directory structure of the site. And you don't need to keep track of how many dot dots, and how many slashes, and all that sort of stuff. You just know that that first slash is always for the root, and then you do the rest of the URL based on the directory structure of your site.

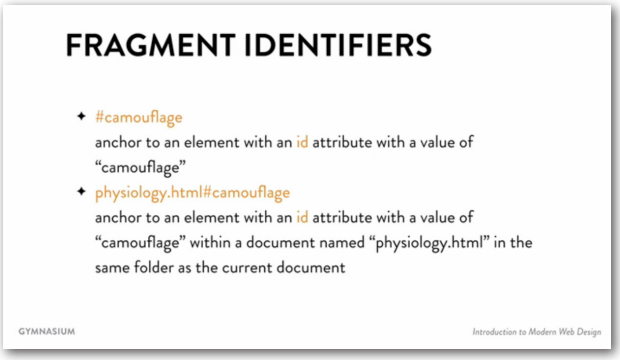
Now, another form of absolute URL actually includes the domain and the protocol that are being used to reference the file. So if I say http://aaron-gustafson.com/i-heart-cuttlefish/physiology.html, that is going to point to physiology.html within the i-heart-cuttlefish subfolder within the root of aaron-gustafson.com. And it's referencing it via http as the protocol.

Similarly, I can reference google.com with https, which is secure http. And that would point to whatever the index file that's defined by the server. That might be index.html. It might be index.php, default.asp. All web servers give us a way of defining what that index file is. And whatever that file is named, that's what will be served if you just went to the domain by itself. By default, that's typically index.html.

It's worth noting that in some cases, you may see a URL that starts with two slashes. This is a particularly odd one, and it's something that's only just starting to become more prevalent. This URL points to the physiology.html file within the i-heart-cuttlefish subfolder of aaron-gustafson.com, but it doesn't define the protocol for accessing that URL. The protocol is left off and is simply implied by the protocol of the existing page.

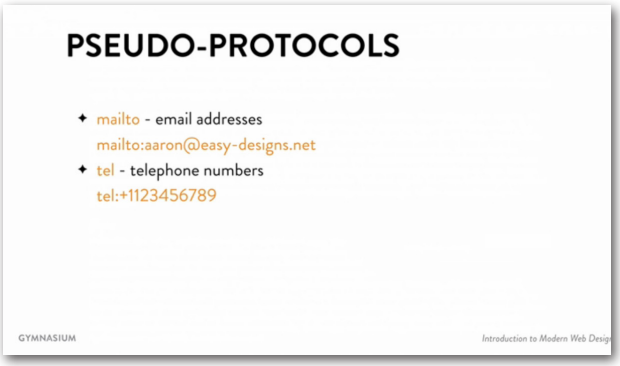
So if the existing page that is linking to this particular URL is under https, so it's a secure server connection, then it would request this file over https. If it's a standard http request, then this URL would be requested over http. Again, you don't see this all that often, but it is starting to crop up, which is why I wanted to point it out.

Now, the last kind of URL I want to talk about are fragment identifiers. Fragment identifiers point to an ID on the page. Now, if you remember, earlier on I showed an instance of an h1 with an id attribute. Now, the id attribute has a value. And if that value were to match the term after the hashtag in a URL, then the browser would actually anchor to that element. It would scroll down the page until that element was at the top of the page in order to bring focus there.



So the first example here, we see #camouflage. In the second example, we see #camouflage within the physiology.html document. Now, fragment identifiers can be used with virtually any URL, so we could even extend it to being used with a fully qualified absolute URL here to physiology.html within i-heart-cuttlefish on the aaron-gustafson.com domain. So if I boot up a browser and show typing in that URL, hit Enter, what you'll see is I'll go to that page and it'll immediately jump down to it. That's pretty awesome.

Now, there are two other kinds of links that I want to bring up really quickly, and that's mailto and tel links. These can be used for pointing out email addresses or telephone numbers, and browsers automatically will trigger your email client to open if you click on a mailto link. If your browser is on a device that supports making a phone call, clicking the tel link will actually prompt the user to dial that number.



Now that we understand all of the different URL types, we'll use most of these in the context of the A element, which we've seen a little bit in a couple of examples already. So the A element is used for links within the document body. So here we have a simple example—"the cuttlebone" and I want to link that to a section about the cuttlebone on another page. Since that section in the other page exists on the physiology.html page, I'm going to go ahead and insert an anchor with an href pointing to physiology.html, and anchoring to "the cuttlebone" ID.

Now, here's a second example from the footer of my site. The text content of the site was lovingly borrowed from the Wikipedia article on cuttlefish on 15 August 2014. I've got my nice little time element in there, and so on and so forth. And I've got the Wikipedia article on cuttlefish and also the licensing of the Creative Commons Attribution ShareAlike license. I can link both of these also using the anchor element to point to the Wikipedia article, or to the text of the Creative Commons Attribution license.

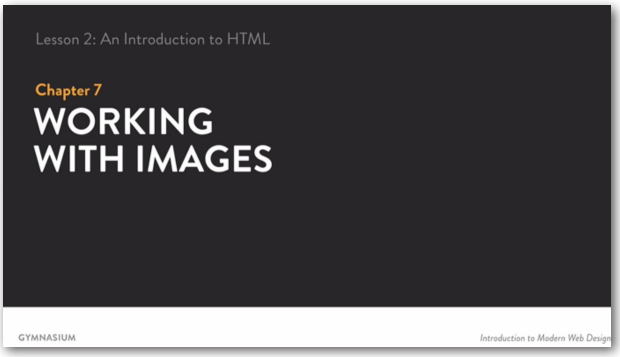
Now, we can also—as we saw earlier with the canonical example in one of the earlier chapters—we can link to other pages within the head of the document using the link element. We saw that with rel=canonical. So for the canonical information about this particular site, we can also use it to link to alternate content. This is an example from my blog, which links to an XML file for the atom feed for my site.

Now, the last one I want to go over is the image element, which is used to embed images, obviously, into the page. And just as you would suspect, and as we've seen in a couple of examples already, you use the source attribute on the img tag to say where that image is.

So in this chapter, we've talked about all the different types of URLs, how to create links to other pages, and also how to create references to other files that we want to make available to our users when they view our page. Now in the next chapter, I'm going to take a deeper dive into images, specifically looking at traditional image usage, as well as responsive images, which will be helpful in dealing with multiple screen sizes and resolutions and the like. I'll see you in a few.

WORKING WITH IMAGES

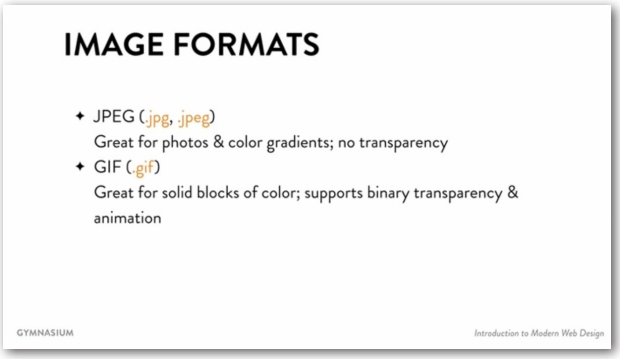
All right, welcome back. In this chapter, we're going to talk a little bit more about working with images. Now, we've seen quite a few examples of the image element already. And we know that img is used to embed image content. We've seen that a bunch. Here we have a brief recap for you.

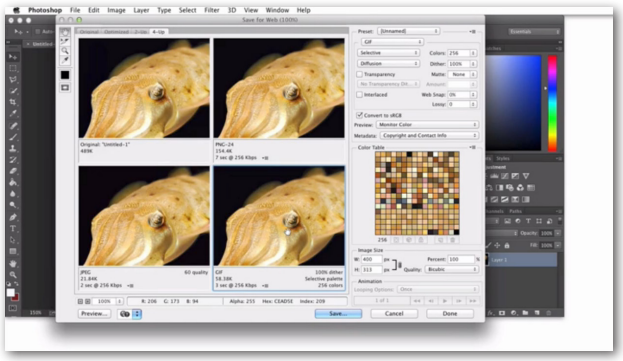


You've got the img element with a source attribute pointing to where the image exists, and then the alt attribute. Now, it's worth noting that the alt attribute is required as alternative text for the image, but if the image in question is actually more presentational, but it doesn't make sense to include it using the style or something like that, then you can use an empty alt attribute as you see here. So alt equals, and then empty quotes.

The reason to do this, apart from being concerned about having a valid HTML document, is that a screen reader, when it comes to an image, will actually read out the word "image" if there is no alt text. If the alt text is empty, it will simply skip over it. If there is alt text, it will read the alternative text instead of reading the word "image."

Now, when using images, it's really important for you to choose the best and most appropriate image type for each context. So for instance, if you have something that's a photo, or has a lot of color gradients, and doesn't require transparency, a JPEG may be the most appropriate image format for you to use, because of the way it compresses the image down.





If, however, you have something with solid blocks of color that requires just some binary transparency, in other words, either fully transparent or fully opaque, then perhaps a GIF is the way to go. Another option would be an 8-bit PNG, which is like a GIF, but has a little bit better compression. Or if you need to have alpha transparency, in other words you need transparency that is not simply on or off, but actually is graded, then you might want to look at using a 24-bit PNG. They're really good for photos and gradients, but they can be a bit larger than a JPEG. A newer format that we're starting to see is the WebP. Not all browsers support it, but it's like a 24-bit PNG, but it's a little bit smaller file size.

Now, if you're using Photoshop as your photo editor, you can actually choose how you export the photos and look at the various ways that it compresses them down. You do that by choosing Save for Web. And then it'll give you a four-up grid of different images. And you can choose the different image formats and see how big they are in terms of file size, as well as how long they would take to download over particular speeds of connection. This can be really useful when trying to figure out the best way to save the image.

If you look in the lower left-hand corner, you can see that the JPEG compresses a bit smaller. And I can even adjust the quality to be a little bit better in order to make the image look sharper. Now, in the lower right-hand corner, I can turn it into a GIF. And that GIF is going to be a little bit more pixely because it only has a limited number of colors available to it. And because it's a photo, it's going to be a little bit bigger than a JPEG is. But both of them are a lot smaller than the 24-bit PNG is.

There are a lot of settings that you can adjust within Photoshop in order to figure out the best possible solution. After all the tweaking, it looks like the JPEG is the best choice at 21.84K. And it'll only take 9 seconds to download, even over a 28.8 kilobits per second connection.

Now, if you want to be able to export the WebP format, there is an option for that. You can download an exporter from telegraphics.com.au. And you'll be able to export directly from Photoshop into the WebP format. But just, again, be aware that WebP is not universally supported at this time, so it may not be the optimum image format to use right now.

Web designer Jenn Lukas has put together an amazing talk available at this URL that you should take a look at. I highly recommend that you check it out.

Now, earlier on in this lesson, I mentioned that HTML as a language has continued to evolve, and that it has mechanisms that allow it to continue to evolve. Now, each of the different specs evolves over a different time period. So HTML is sometimes slower, sometimes faster. CSS runs on its own track, JavaScript on its own track. Everything doesn't move in lockstep. It kind of moves at its own pace. But each of these languages continues to evolve.

And so as part of that evolution, we've had the creation of adaptive images, which are for use within a responsive context. So here we have a standard way of embedding an image. Well, as part of the picture spec for responsive images within HTML5, we can define more options that apply in different contexts.

So first we have the source attribute, which is the default. It's the fallback. Browsers that are older, that don't understand the new attributes from HTML5, will ignore those attributes because browsers ignore what they don't understand. And they'll only see the src, the source attribute, and use that image. So we want to choose the smallest image for that. So here you see images/cuttlefish-sm for small cuttlefish and then dot jpg.

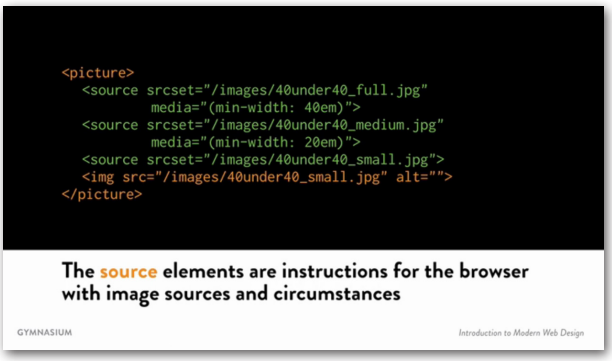
The next attribute we have is the srcset attribute. And what the srcset attribute lets us do is define multiple images for use in different contexts. So you start with the URL for the image, followed by a space, and then the width that that image is. So in this case, cuttlefish.jpg is 800 pixels wide. So I say images/cuttlefish.jpg space 800w. And then I can define an alternate image by putting the comma in there and saying, images/cuttlefish-sm.jpg 400w, because it's 400 pixels wide. With these instructions, a browser will choose the most appropriate image given the available screen real estate.

Now, I can further define how these images should be chosen using the sizes attribute. With sizes, I can actually use a media query. And we'll talk a bit more about media queries in the CSS lessons. But I can use a media query and say that when there is a minimum width of 800 pixels, the image will be displayed at 400 pixels wide. Otherwise, it will be at 50%. So that's basically what that little bit inside of the sizes means.

And what that does is it provides a set of instructions to the browser to inform it how the image is going to be used so that the browser can make the most appropriate decision as to which image should be loaded in which context. So I'm going to hop over to the browser really quick just to show you a video of how this ends up working. Here we see we've got the image with the cuttlefish small JPEG in it. And then I'm going to slowly increase the width of the browser, and you'll see that it will swap out which image is being used.

So now you see it has switched from cuttlefish small to simply cuttlefish.jpg. And if you mouse over it, you can see that the natural size is 800 by 625, but it's only being displayed at 120 by 94. Now, the reason that it's using this higher resolution image is that I'm actually capturing this on a Retina MacBook Pro, so the higher resolution screen is recognized by the browser, and it swaps in the most appropriate image given the context.

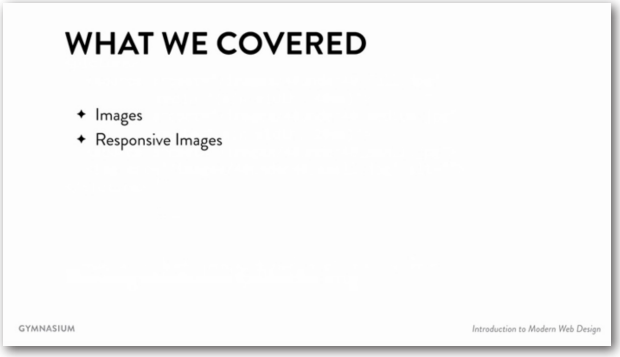
Now, I mentioned that srcset and sizes are part of the picture spec. The picture spec actually came out of the development of the picture element, which is used similarly to srcset and sizes, but specifically for art direction of images. Here we see a sample picture element. We have the actual containing element, the opening and closing picture tags. And then within that, we have multiple sources that are available to us.



And each of those contain their own source sets, so you could have multiple different resolutions in here, and then a media query that's associated with them. So you could even define an image specifically for print if you wanted to. And the way that picture works is that the browser will choose the first image that matches the given context.

So you start with the largest one first. In this case, if the minimum width is 40 ems. In the case of the second one, if the minimum width is 20 ems. And then the third one is simply the default source that should be used by the picture element, which is the small version.

And then last but not least, you'll notice we've got an image element in there. This is actually a requirement. And this is the fallback, because as I've said multiple times, browsers ignore what they don't understand. So that image element becomes a fallback. If the browser understands the picture element, the image gets ignored. But if the browser does not understand the picture element, then that image element actually gets exposed and the picture element's ignored.

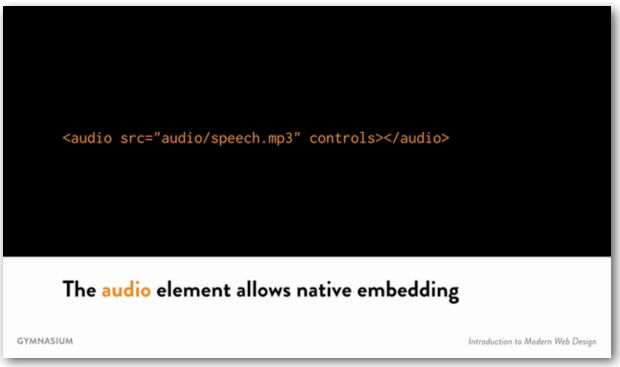


All right. So this wraps up this chapter. And in this chapter, we talked about images, image formats, and the like. And we also talked about using images in responsive contexts. And we'll talk more about responsive design in the CSS lessons. In the next chapter, we're going to talk about other media, specifically audio and video.

WORKING WITH OTHER MEDIA

Welcome back. In this chapter, we're going to be talking about working with other media—in other words, audio and video files. Now the audio element is used to embed audio content in our page. This is a new element that's part of HTML5, and all modern browsers support it. Here we see an example of the audio element with a single source. Browsers that understand the audio element will actually create a player for the audio file and allow somebody to play it, pause it, scrub through it, etc. especially since we have the controls attribute on there.

The controls attribute tells the browser that we want those controls to be available to our users. If you don't include that attribute, the controls would not be displayed. Like images, there are multiple formats for audio, there's WebM, which is an open source audio format, there's MP3, which is licensed, there's Ogg Vorbis, and then there's AAC. Now, it will be great if there was one solution that you could choose and you know is going to work universally well across browsers; unfortunately that doesn't really happen.



If you look at the Mozilla developer network, you can see this table and all of the information about which formats are supported in which browsers, but it's not super consistent. So for this reason, we actually need to define multiple potential sources for the audio. This is where the source element comes in. You'll remember the source element from the picture element in the last chapter.

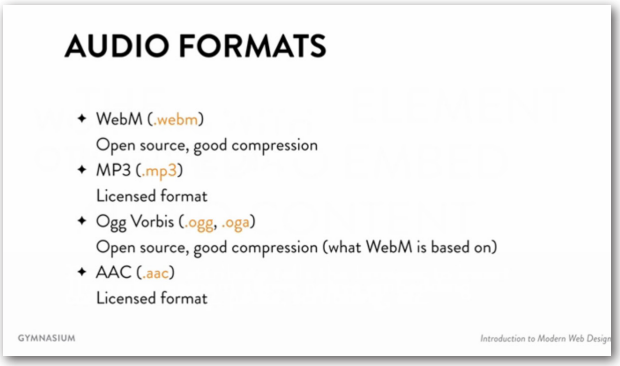
It works in very much the same way. Here, we've defined two different sources, one for an MP3 version and one for an Ogg Vorbis version, or Ogg Audio.

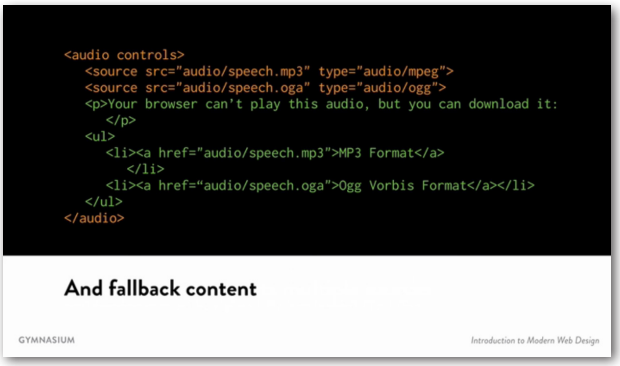
The browser will pick the most appropriate version to it and play that for the user. Of course some older browsers don't support the audio element. So for those browsers, we want to provide some sort of fallback content. We can do that simply by putting the alternate content within the audio element. If the audio element is supported, that content will be ignored. If the audio element isn't supported, the audio element will be ignored, but the content here in green would be exposed, giving direct access to download either the MP3 or the Ogg Vorbis file.

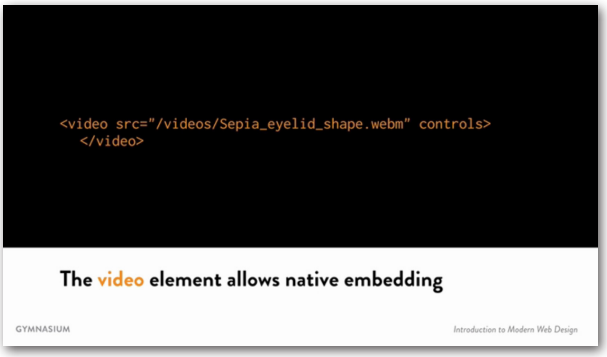
Similar to the audio element is the video element which we use for embedding video content on the page. Here we have a simple video element pointing to a video of a cuttlefish eye. Again, we've got the controls attribute that allows us to indicate whether the controls should be displayed or not. And as with audio, video also has multiple formats available, and there's not really consistency in terms of which one is supported where. And there's some added complexity in that Safari on the desktop supports different formats than Safari on mobile.

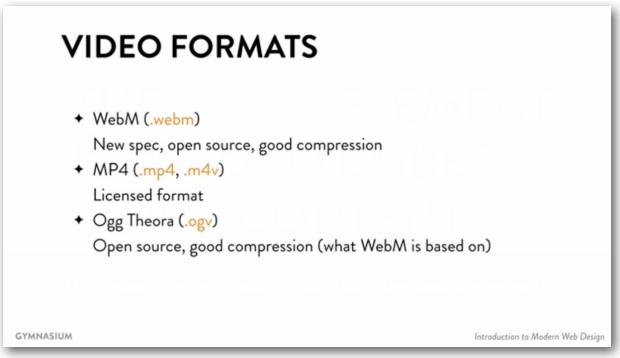
Just like with audio, we can provide alternate video sources using source attributes within the video element itself. First we have the MP4 version, which should be put first because the first match wins and older Apple devices and Internet Explorer will pick up the MP4 version. Most other modern browsers will pick up the WebM version, so we want to put that second. If you need to support older versions of Firefox, you might consider also using the Ogg Theora format and putting that third.

Because that one's going to be a little bit bigger than the WebM file. So again, you want to put them in order of priorities so that the first match gets picked up. But you also want to ensure that if there are multiple potential matches; the smallest one is first so that that gets picked up and downloaded rather than the larger file that's also supported. It's worth noting that the video element also supports a poster attribute that lets you define an image that should be shown before the video is played.



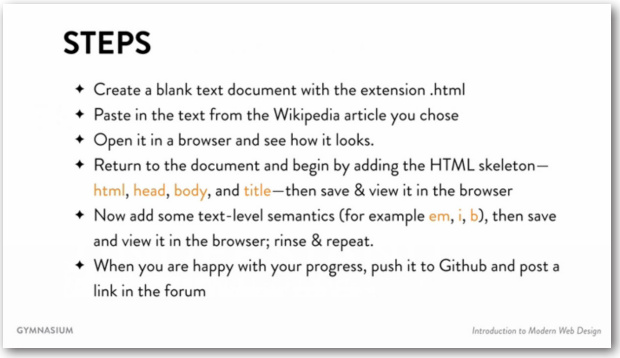






You can also do things like control whether the browsers should preload the video file or not. This controls the bandwidth usage for your web page. And again, as is becoming a bit of a mantra, browsers ignore what they don't understand, so we can provide fallback content for the video element in the form of an image and links to actually download those video files. And that does it for our chapter on audio and video. This has been just a brief introduction to the topics. There's a lot more to explore, but I'm going to leave that to you to do on your own time.

What I am going to have you do, however, is assignment number 3, which is to take quiz number 1, which will test your knowledge of HTML. I also want you to do another assignment, in that I want you to begin marking up the content of your pages. The way to do this would be to create a blank text document with an HTML extension, paste the text from the Wikipedia article you chose into that document, and open it in a browser and see how it looks. Then return to the document and begin adding different elements like html, head, body, and title. Save it, and view that page in the browser and see how it's changed.



Now go back and add some text-level semantics, for example maybe emphasis or i or b or some links, and rinse and repeat. I want you to do this and continue refining the document to make it better and better and keep looking at the browser and see how it's changed. When you're happy with your file, go ahead and push that up to Github and post a link in the forum. Your Github repository should look a little bit something like this, where you have an index.html file in it.

And the HTML file will probably end up looking something along the lines of this. I look forward to seeing you in the next lesson, where we'll dive a little bit deeper into HTML5 block level elements and some of the other options available to us.