Introduction to HTML

At its heart, HTML is a fairly simple language made up of elements, which can be applied to pieces of text to give them different meaning in a document (Is it a paragraph? Is it a bulleted list? Is it part of a table?), structure a document into logical sections (Does it have a header? Three columns of content? A navigation menu?), and embed content such as images and videos into a page. This module will introduce the first two of these and introduce fundamental concepts and syntax you need to know to understand HTML.

Prerequisites

Before starting this module, you don't need any previous HTML knowledge, but you should have at least basic familiarity with using computers and using the web passively (i.e., just looking at it and consuming content). You should have a basic work environment set up (as detailed in Installing basic software), and understand how to create and manage files (as detailed in Dealing with files). Both are parts of our Getting started with the web complete beginner's module.

> **Note**: If you are working on a computer/tablet/other devices that doesn't let you create your own files, you can try out (most of) the code examples in an online coding program such as JSBin or Glitch.

Guides

This module contains the following articles, which will take you through all the basic theory of HTML and provide ample opportunity for you to test out some skills.

Getting started with HTML

Covers the absolute basics of HTML, to get you started — we define elements, attributes, and other important terms, and show where they fit in the language. We also show how a typical HTML page is structured and how an HTML element is structured, and explain other important basic language features. Along the way, we'll play with some HTML to get you interested!

What's in the head? Metadata in HTML

The head of an HTML document is the part that **is not** displayed in the web browser when the page is loaded. It contains information such as the page `<title>`, links to CSS (if you want to style your HTML content with CSS), links to custom favicons, and metadata (data about the HTML, such as who wrote it, and important keywords that describe the document).

HTML text fundamentals

One of HTML's main jobs is to give text meaning (also known as semantics), so that the browser knows how to display it correctly. This article looks at how to use HTML to break up a block of text into a structure of headings and paragraphs, add emphasis/importance to words, create lists, and more.

Creating hyperlinks

Hyperlinks are really important — they are what makes the web a web. This article shows the syntax required to make a link and discusses best practices for links.

Advanced text formatting

There are many other elements in HTML for formatting text that we didn't get to in the HTML text fundamentals article. The elements here are less well-known, but still useful to know about. In this article, you'll learn about marking up quotations, description lists, computer code and other related text, subscript and superscript, contact information, and more.

Document and website structure

As well as defining individual parts of your page (such as "a paragraph" or "an image"), HTML is also used to define areas of your website (such as "the header," "the navigation menu," or "the main content column.") This article looks into how to plan a basic website structure and how to write the HTML to represent this structure.

Debugging HTML

Writing HTML is fine, but what if something goes wrong, and you can't work out where the error in the code is? This article will introduce you to some tools that can help.

Assessments

The following assessments will test your understanding of the HTML basics covered in the guides above.

Marking up a letter

We all learn to write a letter sooner or later; it is also a useful example to test out text formatting skills. In this assessment, you'll be given a letter to mark up.

Structuring a page of content

This assessment tests your ability to use HTML to structure a simple page of content, containing a header, a footer, a navigation menu, main content, and a sidebar.

See also

Web literacy basics 1

An excellent Mozilla foundation course that explores and tests a lot of the skills talked about in the *Introduction to HTML* module. Learners get familiar with reading, writing, and participating on the web in this six-part module. Discover the foundations of the web through production and collaboration.

Feedback

Help us improve our guides and tutorials like this one by taking our survey here.

Last modified: Apr 17, 2020, by MDN contributors

Getting started with HTML

Overview: Introduction to HTML Next

In this article we cover the absolute basics of HTML. To get you started, this article defines elements, attributes, and all the other important terms you may have heard. It also explains where these fit into HTML. You will learn how HTML elements are structured, how a typical HTML page is structured, and other important basic language features. Along the way, there will be an opportunity to play with HTML too!

Prerequisites:	Basic computer literacy, basic software installed, and basic knowledge of working with files.
Objective:	To gain basic familiarity with HTML, and practice writing a few HTML elements.

What is HTML?

HTML (Hypertext Markup Language) is not a programming language. It is a *markup language* that tells web browsers how to structure the web pages you visit. It can be as complicated or as simple as the web developer wants it to be. HTML consists of a series of elements, which you use to enclose, wrap, or *mark up* different parts of content to make it appear or act a certain way. The enclosing tags can make content into a hyperlink to link to another page, italicize words, and so on. For example, consider the following line of text:

```
My cat is very grumpy
```

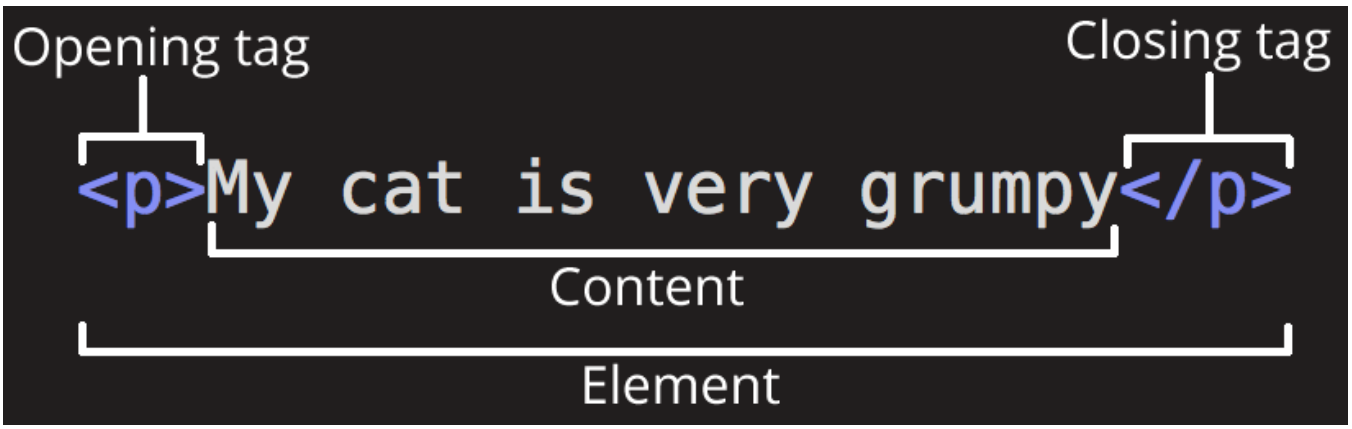
If we wanted the text to stand by itself, we could specify that it is a paragraph by enclosing it in a paragraph (`<p>`) element:

```
<p>My cat is very grumpy</p>
```

Note: Tags in HTML are case-insensitive. This means they can be written in uppercase or lowercase. For example, a `<title>` tag could be written as `<title>`, `<TITLE>`, `<Title>`, `<TiTlE>`, etc., and it will work. However, it is best practice to write all tags in lowercase for consistency, readability, and other reasons.

Anatomy of an HTML element

Let's explore our paragraph element a bit further:



The main parts of our element are:

- **The opening tag:** This consists of the name of the element (in this case, p), wrapped in opening and closing **angle brackets**. This states where the element begins or starts to take effect — in this case where the start of the paragraph is.
- **The closing tag:** This is the same as the opening tag, except that it includes a forward slash before the element name. This states where the element ends — in this case where the end of the paragraph is. Failing to include a closing tag is a common beginner error and can lead to strange results.
- **The content:** This is the content of the element, which in this case is just text.
- **The element:** The opening tag plus the closing tag plus the content equals the element.

Active learning: creating your first HTML element

Edit the line below in the *Input* area by wrapping it with the tags `<em>` and `</em>` (put `<em>` before it to *open the element*, and `</em>` after it, to *close the element*) — this should give the line italic emphasis! You'll be able to see your changes update live in the *Output* area.

If you make a mistake, you can always reset it using the *Reset* button. If you get really stuck, press the *Show solution* button to see the answer.

Live output

This is my text.

Editable code

Press Esc to move focus away from the code area (Tab inserts a tab character).

```
This is my text.
```

Reset | Show solution

Nesting elements

Elements can be placed within other elements too — this is called **nesting**. If we wanted to state that our cat is **very** grumpy, we could wrap the word "very" in a `<strong>` element, which means that the word is to be strongly emphasized:

```
1 <p>My cat is <strong>very</strong> grumpy.</p>
```

You do however need to make sure that your elements are properly nested: in the example above, we opened the `p` element first, then the `strong` element, therefore we have to close

the `strong` element first, then the `p`. The following is incorrect:

```
<p>My cat is <strong>very grumpy.</p></strong>
```

The elements have to open and close correctly, so they are clearly inside or outside one another. If they overlap like above, then your web browser will try to make a best guess at what you were trying to say, and you may well get unexpected results. So don't do it!

Block versus inline elements

There are two important categories of elements in HTML which you should know about. They are block-level elements and inline elements.

- Block-level elements form a visible block on a page — they will appear on a new line from whatever content went before it, and any content that goes after it will also appear on a new line. Block-level elements tend to be structural elements on the page that represent, for example, paragraphs, lists, navigation menus, footers, and so on. A block-level element wouldn't be nested inside an inline element, but it might be nested inside another block-level element.
- Inline elements are those that are contained within block-level elements and surround only small parts of the document's content, not entire paragraphs and groupings of content. An inline element will not cause a new line to appear in the document; they would normally appear inside a paragraph of text, for example an `<a>` element (hyperlink) or emphasis elements such as `<em>` or `<strong>`.

Take the following example:

```
<em>first</em><em>second</em><em>third</em>

<p>fourth</p><p>fifth</p><p>sixth</p>
```

`<em>` is an inline element, so as you can see below, the first three elements sit on the same line as one another with no space in between. On the other hand, `<p>` is a block-level element, so each element appears on a new line, with space above and below each (the spacing is due to default CSS styling that the browser applies to paragraphs).

firstsecondthird

fourth

fifth

sixth

Note: HTML5 redefined the element categories in HTML5: see Element content categories. While these definitions are more accurate and less ambiguous than the ones that went before, they are a lot more complicated to understand than "block" and "inline", so we will stick with these throughout this topic.

Note: The terms "block" and "inline", as used in this topic, should not be confused with the types of CSS boxes with the same names. While they correlate by default, changing the CSS display type doesn't change the category of the element and doesn't affect which elements it can contain and which elements it can be contained in. One of the reasons why HTML5 dropped these terms was to prevent this rather common confusion.

Note: You can find useful reference pages that include lists of block and inline elements — see Block-level elements and Inline elements.

Empty elements

Not all elements follow the above pattern of an opening tag, content, and a closing tag. Some elements consist only of a single tag, which is usually used to insert/embed something in the document at the place it is included. For example, the `<img>` element embeds an image file onto a page in the position it is included in:

```
1 <img src="https://raw.githubusercontent.com/mdn/beginner-html-site/gh-pag
```

This would output the following on your page:

> **Note**: Empty elements are also sometimes called *void elements*.

Attributes

Elements can also have attributes, which look like this:

Attributes contain extra information about the element that you don't want to appear in the actual content. In this case, the `class` attribute allows you to give the element an identifying name, that can be used later to target the element with style information and other things.

An attribute should have:

- A space between it and the element name (or the previous attribute, if the element already has one or more attributes).
- The attribute name, followed by an equal sign.
- An attribute value, with opening and closing quote marks wrapped around it.

Active learning: Adding attributes to an element

Another example of an element is `<a>` — this stands for "anchor" and will make the piece of text it wraps around into a hyperlink. This can take a number of attributes, but several are as follows:

- **`href`**: This attribute's value specifies the web address that you want the link to point to; where the browser navigates to when the link is clicked. For example, `href="https://www.mozilla.org/"`.
- **`title`**: The `title` attribute specifies extra information about the link, such as what page is being linked to. For example, `title="The Mozilla homepage"`. This will appear as a tooltip when the element is hovered over.
- **`target`**: The `target` attribute specifies the browsing context that will be used to display the link. For example, `target="_blank"` will display the link in a new tab. If you want to display the link in the current tab, just omit this attribute.

Edit the line below in the *Input* area to turn it into a link to your favorite website.

1. First, add the `<a>` element.
2. Second, add the `href` attribute and the `title` attribute.
3. Lastly, specify the `target` attribute to open the link in the new tab.

You'll be able to see your changes update live in the *Output* area. You should see a link that when hovered over displays the value of the `title` attribute, and when clicked, navigates to the web address in the `href` attribute. Remember that you need to include a space between the element name, and each attribute.

If you make a mistake, you can always reset it using the *Reset* button. If you get really stuck, press the *Show solution* button to see the answer.

Live output

A link to my favorite website.

Editable code

Press Esc to move focus away from the code area (Tab inserts a tab character).

```
<p>A link to my favorite website.</p>
```

Reset Show solution

Boolean attributes

You'll sometimes see attributes written without values — this is perfectly allowed. These are called Boolean attributes, and they can only have one value, which is generally the same as the attribute name. As an example, take the `disabled` attribute, which you can assign to form input elements, if you want them to be disabled (greyed out) so the user can't enter any data in them.

```
<input type="text" disabled="disabled">
```

As shorthand, it is perfectly allowable to write this as follows (we've also included a non-disabled form input element for reference, to give you more of an idea what is going on):

```
<!-- using the disabled attribute prevents the end user from entering te
<input type="text" disabled>

<!-- The user can enter text into the follow input, as it doesn't contair
<input type="text">
```

The above HTML will give you a rendered output as follows:

Omitting quotes around attribute values

When you look around the World Wide Web, you'll come across a number of strange markup styles, including attribute values without quotes. This is allowable in certain circumstances, but will break your markup in others. For example, if we revisit our link example from earlier, we could write a basic version with only the `href` attribute, like this:

```
1 <a href=https://www.mozilla.org/>favorite website</a>
```

However, as soon as we add the `title` attribute in this style, things will go wrong:

```
1 <a href=https://www.mozilla.org/ title=The Mozilla homepage>favorite webs
```

At this point the browser will misinterpret your markup, thinking that the `title` attribute is actually three attributes — a title attribute with the value "The", and two Boolean attributes, `Mozilla` and `homepage`. This is obviously not what was intended, and will cause errors or unexpected behavior in the code, as seen in the live example below. Try hovering over the link to see what the title text is!

favorite website favorite website

Our advice is to always include the attribute quotes — it avoids such problems, and results in more readable code too.

Single or double quotes?

In this article you'll notice that the attributes are all wrapped in double quotes. You might however see single quotes in some people's HTML code. This is purely a matter of style, and you can feel free to choose which one you prefer. Both the following lines are equivalent:

```
<a href="http://www.example.com">A link to my example.</a>

<a href='http://www.example.com'>A link to my example.</a>
```

You should however make sure you don't mix them together. The following will go wrong!

```
<a href="http://www.example.com'>A link to my example.</a>
```

If you've used one type of quote in your HTML, you can include the other type of quote inside your attribute values without causing any problems:

```
<a href="http://www.example.com" title="Isn't this fun?">A link to my exa
```

However if you want to include a quote, within the quotes where both the quotes are of the same type (single quote or double quote), you'll have to use HTML entities for the quotes. For example, this will break:

```
<a href='http://www.example.com' title='Isn't this fun?'>A link to my exa
```

So you need to do this:

```
<a href='http://www.example.com' title='Isn&#39;t this fun?'>A link to my
```

Anatomy of an HTML document

That wraps up the basics of individual HTML elements, but they aren't very useful on their own. Now we'll look at how individual elements are combined to form an entire HTML page:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>My test page</title>
  </head>
  <body>
    <p>This is my page</p>
  </body>
</html>
```

Here we have:

1. `<!DOCTYPE html>`: The doctype. In the mists of time, when HTML was young (about 1991/2), doctypes were meant to act as links to a set of rules that the HTML page had to follow to be considered good HTML, which could mean automatic error checking and other useful things. They used to look something like this:

   ```
   <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
   ```

   However, these days no one really cares about them, and they are really just a historical artifact that needs to be included for everything to work right. `<!DOCTYPE html>` is the shortest string of characters that counts as a valid doctype; that's all you really need to know.

2. `<html></html>`: The `<html>` element. This element wraps all the content on the entire page, and is sometimes known as the root element.
3. `<head></head>`: The `<head>` element. This element acts as a container for all the stuff you want to include on the HTML page, that *isn't* the content you are showing to your page's viewers. This includes things like keywords and a page description that you want

to appear in search results, CSS to style our content, character set declarations, and more. You'll learn more about this in the next article of the series.

4. `<meta charset="utf-8">`: This element specifies the character set for your document to UTF-8, which includes most characters from the vast majority of human written languages. Essentially, it can now handle any textual content you might put on it. There is no reason not to set this, and it can help avoid some problems later.
5. `<title></title>`: The `<title>` element. This sets the title of your page, which is the title that appears in the browser tab the page is loaded in, and is used to describe the page when you bookmark/favorite it.
6. `<body></body>`: The `<body>` element. This contains *all* the content that you want to show to web users when they visit your page, whether that's text, images, videos, games, playable audio tracks, or whatever else.

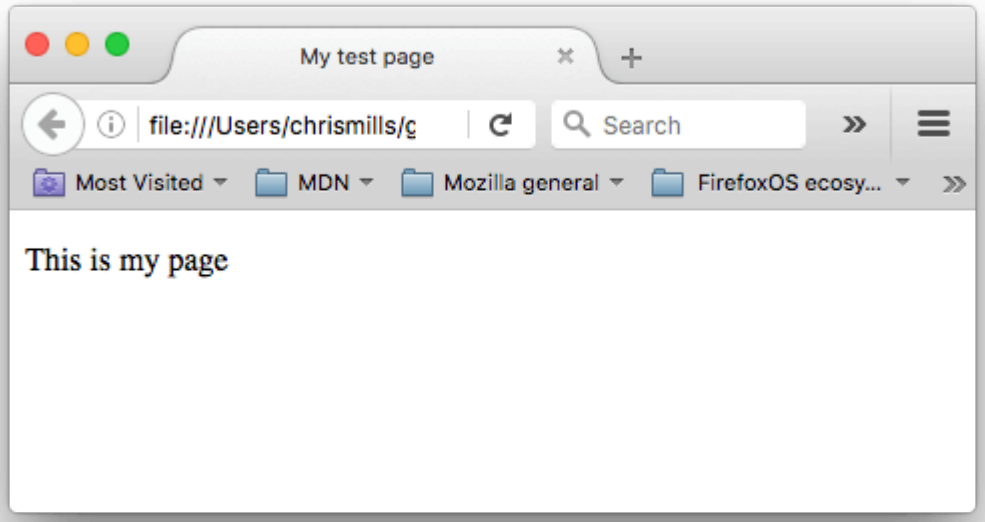
Active learning: Adding some features to an HTML document

If you want to experiment with writing some HTML on your local computer, you can:

1. Copy the HTML page example listed above.
2. Create a new file in your text editor.
3. Paste the code into the new text file.
4. Save the file as `index.html`.

> **Note**: You can also find this basic HTML template on the MDN Learning Area Github repo.

You can now open this file in a web browser to see what the rendered code looks like, and then edit the code and refresh the browser to see what the result is. Initially it will look like this:



So in this exercise, you can edit the code locally on your computer, as outlined above, or you can edit it in the editable sample window below (the editable sample window represents just the contents of the `<body>` element, in this case). We'd like you to have a go at implementing the following tasks:

- Just below the opening tag of the `<body>` element, add a main title for the document. This should be wrapped inside an `<h1>` opening tag and `</h1>` closing tag.
- Edit the paragraph content to include some text about something you are interested in.
- Make any important words stand out in bold by wrapping them inside a `<strong>` opening tag and `</strong>` closing tag.
- Add a link to your paragraph, as explained earlier in the article.
- Add an image to your document, below the paragraph, as explained earlier in the article. You'll get bonus points if you manage to link to a different image (either locally on your computer, or somewhere else on the web).

If you make a mistake, you can always reset it using the *Reset* button. If you get really stuck, press the *Show solution* button to see the answer.

Live output

This is my page

Editable code

Press Esc to move focus away from the code area (Tab inserts a tab character).

```
<p>This is my page</p>
```

Reset Show solution

Whitespace in HTML

In the above examples you may have noticed that a lot of whitespace is included in the code listings — this is not necessary at all; the two following code snippets are equivalent:

```
<p>Dogs are silly.</p>

<p>Dogs        are
         silly.</p>
```

No matter how much whitespace you use inside HTML element content (which can include space characters, but also line breaks), the HTML parser reduces each sequence of

whitespace down to a single space when rendering the code. So why use so much whitespace? The answer is readability — it is so much easier to understand what is going on in your code if you have it nicely formatted, and not just bunched up together in a big mess. In our HTML we've got each nested element indented by two spaces more than the one it is sitting inside. It is up to you what style of formatting you use (how many spaces for each level of indentation, for example), but you should consider formatting it.

Entity references: Including special characters in HTML

In HTML, the characters `<`, `>`, `"`, `'` and `&` are special characters. They are parts of the HTML syntax itself, so how do you include one of these characters in your text? For example, if you really want to use an ampersand or less-than sign, and not have it interpreted as code.

We have to use character references — special codes that represent characters, and can be used in these exact circumstances. Each character reference is started with an ampersand (&), and ended by a semicolon (;).

Literal character	Character reference equivalent
<	<
>	>
"	"
'	'
&	&

The character reference equivalent could be easily remembered because the words it uses can be seen as less than for '<' , quotation for ' " ' and similarly for each. Do checkout the link to the wikipedia page to find more about entity reference. In the example below, you can see two paragraphs, which are talking about web technologies:

```
<p>In HTML, you define a paragraph using the <p> element.</p>

<p>In HTML, you define a paragraph using the &lt;p&gt; element.</p>
```

In the live output below, you can see that the first paragraph has gone wrong, because the browser thinks that the second instance of `<p>` is starting a new paragraph. The second paragraph looks fine, because we have replaced the angle brackets with character references.

In HTML, you define a paragraph using the

element.

In HTML, you define a paragraph using the <p> element.

> **Note**: A chart of all the available HTML character entity references can be found on Wikipedia: List of XML and HTML character entity references. Note that you don't need to use entity references for any other symbols, as modern browsers will handle the actual symbols just fine as long, as your HTML's character encoding is set to UTF-8.

HTML comments

In HTML, as with most programming languages, there is a mechanism available to write comments in the code — comments are ignored by the browser and are invisible to the user, and their purpose is to allow you to include comments in the code to say how your code works, what the different parts of the code do, and so on. This can be very useful if you return to a code base that you've not worked on for a long time, and can't remember what you did — or if you hand your code over for someone else to work on.

To turn a section of HTML content into a comment, you need to wrap it in the special markers `<!--` and `-->`, for example:

```
<p>I'm not inside a comment</p>

<!-- <p>I am!</p> -->
```

As you can see below, the first paragraph appears in the live output, but the second one doesn't.

I'm not inside a comment

Summary

You've reached the end of the article — we hope you enjoyed your tour of the very basics of HTML! At this point you should understand what the language looks like, how it works at a basic level, and be able to write a few elements and attributes. This is the perfect place to be right now, since the subsequent articles of this module will go into some of the things you have already looked at in a lot more detail, and introduce some new concepts of the language. Stay tuned!

> **Note**: At this point, as you start to learn more about HTML, you might also want to start to explore the basics of Cascading Style Sheets, or CSS. CSS is the language you use to style your web pages (e.g., changing the font or colors, or altering the page layout). HTML and CSS go very well together, as you'll soon discover.

See also

- Applying color to HTML elements using CSS

Overview: Introduction to HTML Next

What's in the head? Metadata in HTML

Previous Overview: Introduction to HTML Next

The head of an HTML document is the part that is not displayed in the web browser when the page is loaded. It contains information such as the page `<title>`, links to CSS (if you choose to style your HTML content with CSS), links to custom favicons, and other metadata (data about the HTML, such as the author, and important keywords that describe the document.) In this article we'll cover all of the above and more, in order to give you a good basis for working with markup.

Prerequisites:	Basic HTML familiarity, as covered in Getting started with HTML.
Objective:	To learn about the HTML head, its purpose, the most important items it can contain, and what effect it can have on the HTML document.

What is the HTML head?

Let's revisit the simple HTML document we covered in the previous article:

```
<!DOCTYPE html>
<html>
  <head>
```

```
    <meta charset="utf-8">
    <title>My test page</title>
  </head>
  <body>
    <p>This is my page</p>
  </body>
</html>
```

The HTML head is the contents of the `<head>` element — unlike the contents of the `<body>` element (which are displayed on the page when loaded in a browser), the head's content is not displayed on the page. Instead, the head's job is to contain metadata about the document. In the above example, the head is quite small:

```
<head>
  <meta charset="utf-8">
  <title>My test page</title>
</head>
```

In larger pages however, the head can get quite full. Try going to some of your favorite websites and use the developer tools to check out their head contents. Our aim here is not to show you how to use everything that can possibly be put in the head, but rather to teach you how to use the major elements that you'll want to include in the head, and give you some familiarity. Let's get started.

Adding a title

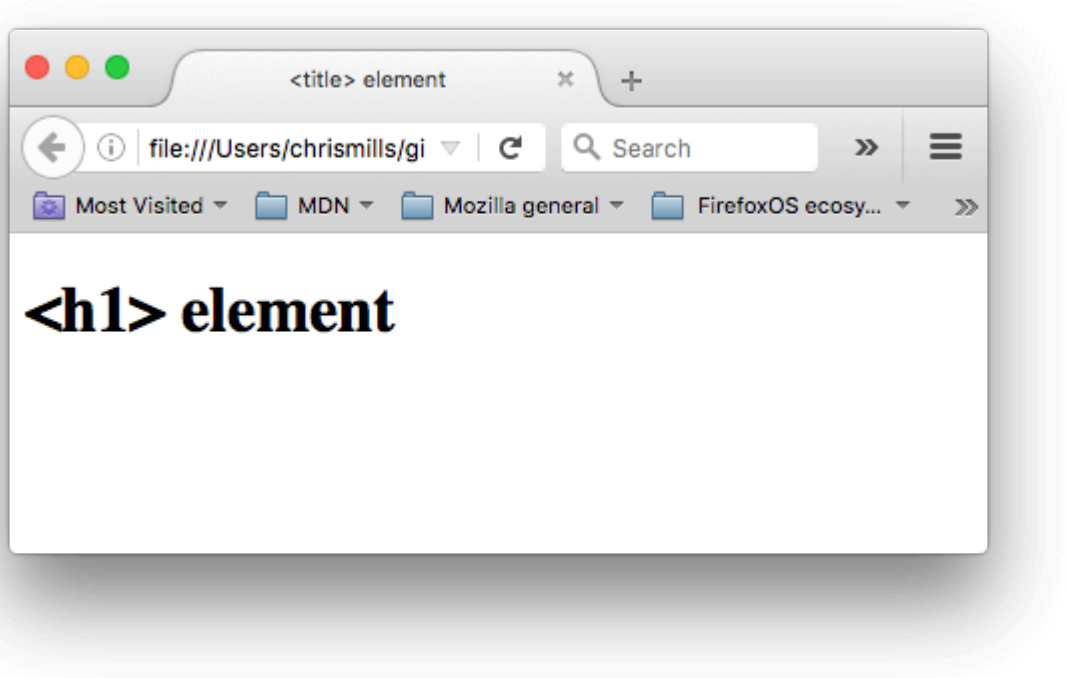
We've already seen the `<title>` element in action — this can be used to add a title to the document. This however can get confused with the `<h1>` element, which is used to add a top level heading to your body content — this is also sometimes referred to as the page title. But they are different things!

- The `<h1>` element appears on the page when loaded in the browser — generally this should be used once per page, to mark up the title of your page content (the story title, or news headline, or whatever is appropriate to your usage.)

- The `<title>` element is metadata that represents the title of the overall HTML document (not the document's content.)

Active learning: Inspecting a simple example

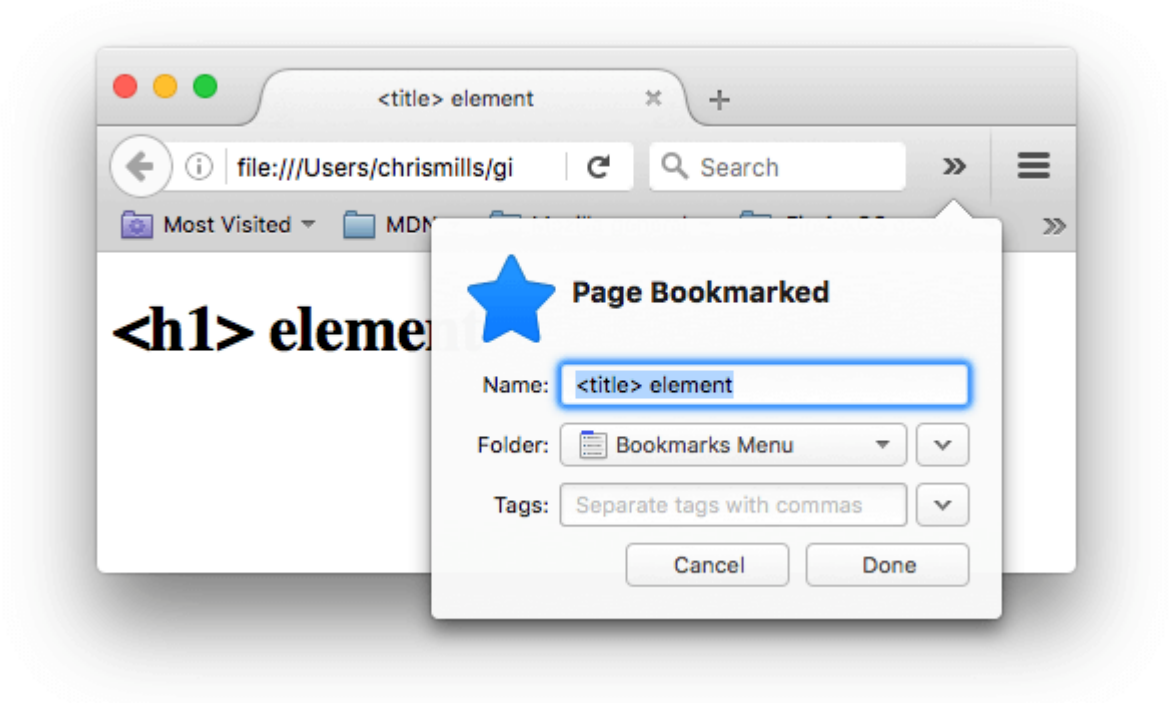
1. To start off this active learning, we'd like you to go to our GitHub repo and download a copy of our title-example.html page. To do this, either
    1. Copy and paste the code out of the page and into a new text file in your code editor, then save it in a sensible place.
    2. Press the "Raw" button on the GitHub page, which causes the raw code to appear (possibly in a new browser tab). Next, choose your browser's *File* > *Save Page As...* menu and choose a sensible place to save the file.
2. Now open the file in your browser. You should see something like this:



   It should now be completely obvious where the `<h1>` content appears, and where the `<title>` content appears!

3. You should also try opening the code up in your code editor, editing the contents of these elements, then refreshing the page in your browser. Have some fun with it.

The `<title>` element contents are also used in other ways. For example, if you try bookmarking the page (*Bookmarks* > *Bookmark This Page* or the star icon in the URL bar in Firefox), you will see the `<title>` contents filled in as the suggested bookmark name.



The `<title>` contents are also used in search results, as you'll see below.

Metadata: the <meta> element

Metadata is data that describes data, and HTML has an "official" way of adding metadata to a document — the `<meta>` element. Of course, the other stuff we are talking about in this article could also be thought of as metadata too. There are a lot of different types of `<meta>` elements that can be included in your page's <head>, but we won't try to explain them all at this stage, as it would just get too confusing. Instead, we'll explain a few things that you might commonly see, just to give you an idea.

Specifying your document's character encoding

In the example we saw above, this line was included:

```
1 <meta charset="utf-8">
```

This element simply specifies the document's character encoding — the character set that the document is permitted to use. `utf-8` is a universal character set that includes pretty much any

character from any human language. This means that your web page will be able to handle displaying any language; it's therefore a good idea to set this on every web page you create! For example, your page could handle English and Japanese just fine:

If you set your character encoding to `ISO-8859-1`, for example (the character set for the Latin alphabet), your page rendering may appear all messed up:

> **Note**: Some browsers (e.g. Chrome) automatically fix incorrect encodings, so depending on what browser you use, you may not see this problem anyway. You should still set an encoding of `utf-8` on your page anyway, to avoid any potential problems in other browsers.

Active learning: Experiment with character encoding

To try this out, revisit the simple HTML template you obtained in the previous section on `<title>` (the title-example.html page), try changing the meta charset value to `ISO-8859-1`, and add the Japanese to your page. This is the code we used:

```
<p>Japanese example: ご飯が熱い。</p>
```

Adding an author and description

Many `<meta>` elements include `name` and `content` attributes:

- `name` specifies the type of meta element it is; what type of information it contains.
- `content` specifies the actual meta content.

Two such meta elements that are useful to include on your page define the author of the page, and provide a concise description of the page. Let's look at an example:

```
<meta name="author" content="Chris Mills">
<meta name="description" content="The MDN Web Docs Learning Area aims to
complete beginners to the Web with all they need to know to get
started with developing web sites and applications.">
```

Specifying an author is beneficial in many ways: it is useful to be able to understand who wrote the page, if you have any questions about the content and you would like to contact them. Some content management systems have facilities to automatically extract page author information and make it available for such purposes.

Specifying a description that includes keywords relating to the content of your page is useful as it has the potential to make your page appear higher in relevant searches performed in search

engines (such activities are termed Search Engine Optimization, or SEO.)

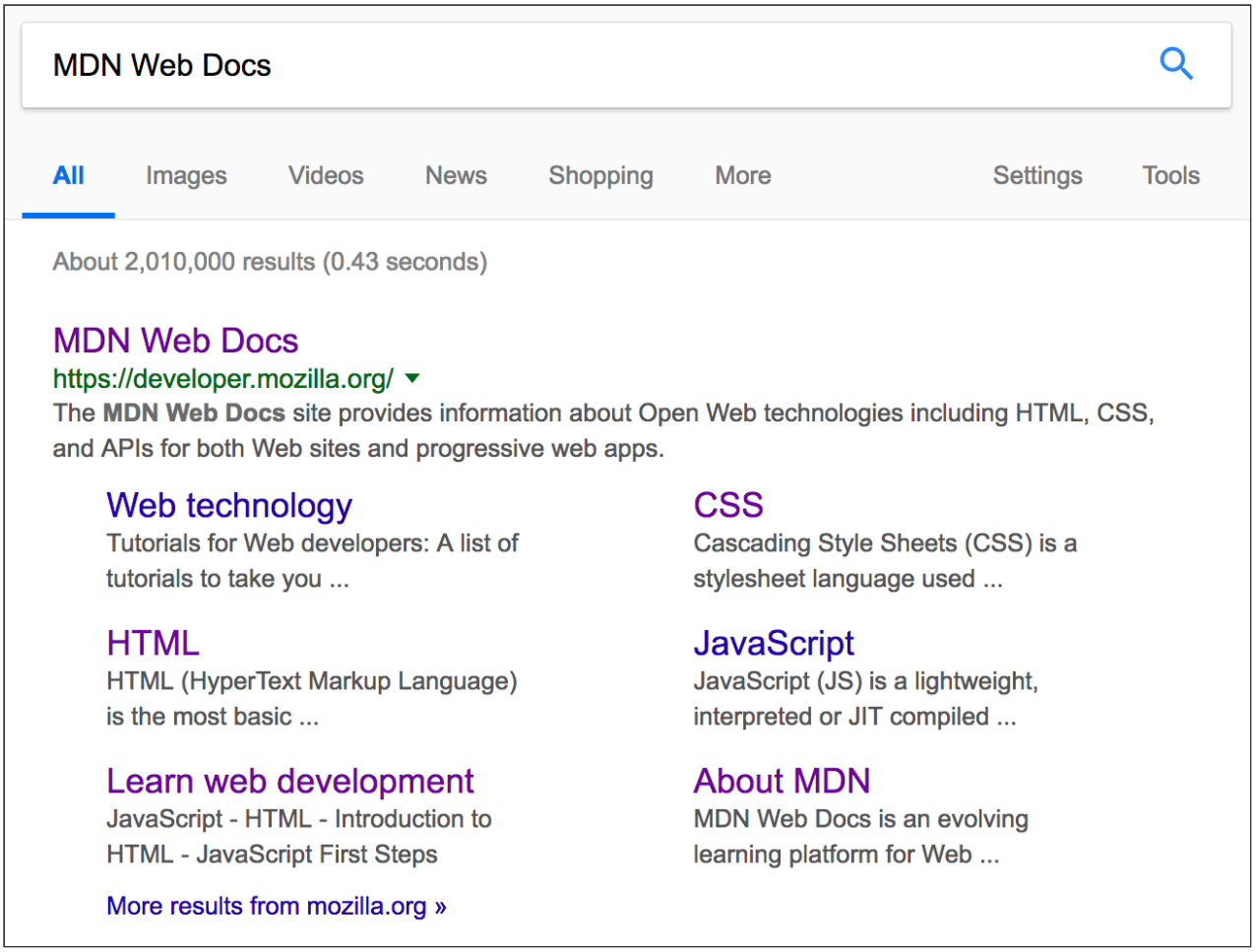
Active learning: The description's use in search engines

The description is also used on search engine result pages. Let's go through an exercise to explore this

1. Go to the front page of The Mozilla Developer Network.
2. View the page's source (Right/`Ctrl` + click on the page, choose *View Page Source* from the context menu.)
3. Find the description meta tag. It will look something like this (although it may change over time):

```
<meta name="description" content="The MDN Web Docs site
  provides information about Open Web technologies
  including HTML, CSS, and APIs for both Web sites and
  progressive web apps.">
```

4. Now search for "MDN Web Docs" in your favorite search engine (We used Google.) You'll notice the description `<meta>` and `<title>` element content used in the search result — definitely worth having!



> **Note**: In Google, you will see some relevant subpages of MDN Web Docs listed below the main homepage link — these are called sitelinks, and are configurable in Google's webmaster tools — a way to make your site's search results better in the Google search engine.

> **Note**: Many `<meta>` features just aren't used any more. For example, the keyword `<meta>` element (`<meta name="keywords" content="fill, in, your, keywords, here">`) — which is supposed to provide keywords for search engines to determine relevance of that page for different search terms — is ignored by search engines, because spammers were just filling the keyword list with hundreds of keywords, biasing results.

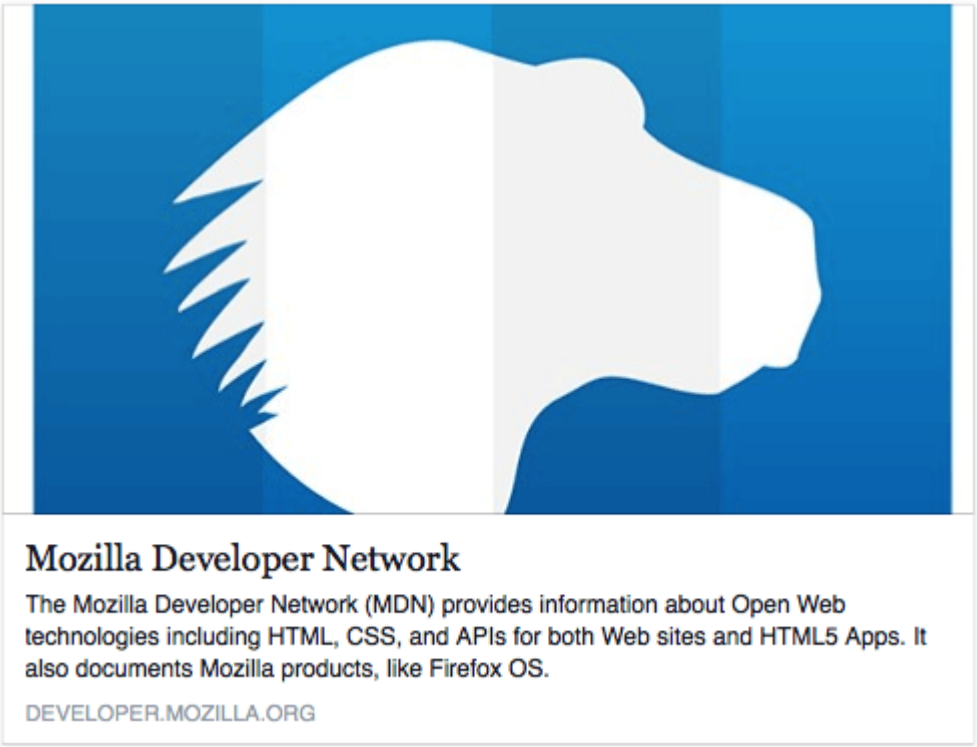
Other types of metadata

As you travel around the web, you'll find other types of metadata, too. A lot of the features you'll see on websites are proprietary creations, designed to provide certain sites (such as social networking sites) with specific pieces of information they can use.

For example, Open Graph Data is a metadata protocol that Facebook invented to provide richer metadata for websites. In the MDN Web Docs sourcecode, you'll find this:

```
<meta property="og:image" content="https://developer.cdn.mozilla.net/stat
<meta property="og:description" content="The Mozilla Developer Network (M
information about Open Web technologies including HTML, CSS, and APIs for
and HTML5 Apps. It also documents Mozilla products, like Firefox OS.">
<meta property="og:title" content="Mozilla Developer Network">
```

One effect of this is that when you link to MDN Web Docs on facebook, the link appears along with an image and description: a richer experience for users.



Twitter also has its own similar proprietary metadata called Twitter Cards, which has a similar effect when the site's URL is displayed on twitter.com. For example:

```
<meta name="twitter:title" content="Mozilla Developer Network">
```

Adding custom icons to your site

To further enrich your site design, you can add references to custom icons in your metadata, and these will be displayed in certain contexts. The most commonly used of these is the **favicon** (short for "favorites icon", referring to its use in the "favorites" or "bookmarks" lists in browsers).

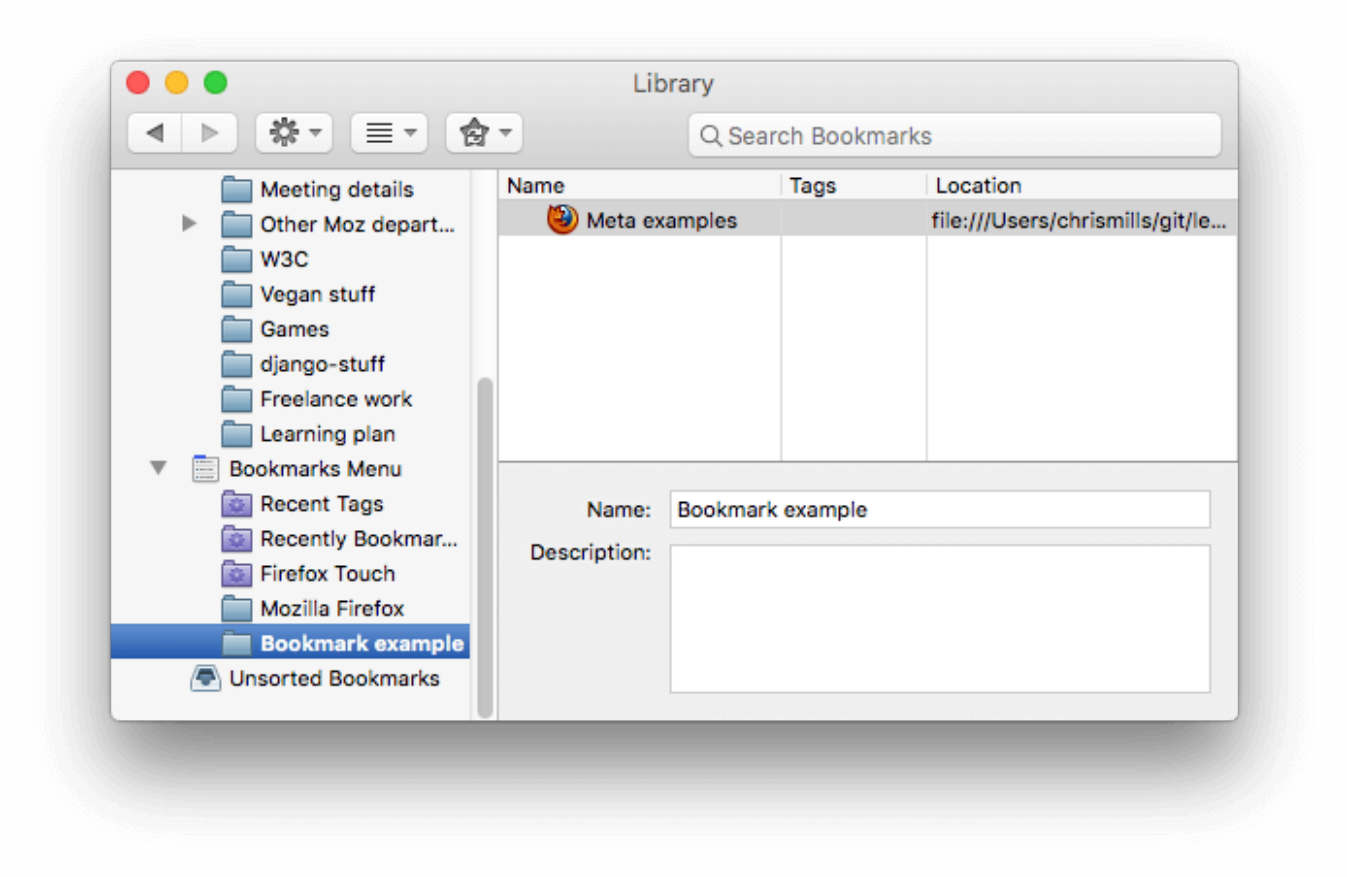
The humble favicon has been around for many years. It is the first icon of this type: a 16-pixel square icon used in multiple places. You may see (depending on the browser) favicons displayed in the browser tab containing each open page, and next to bookmarked pages in the bookmarks panel.

A favicon can be added to your page by:

1. Saving it in the same directory as the site's index page, saved in `.ico` format (most browsers will support favicons in more common formats like `.gif` or `.png`, but using the ICO format will ensure it works as far back as Internet Explorer 6.)
2. Adding the following line into your HTML's `<head>` block to reference it:

```
<link rel="icon" href="favicon.ico" type="image/x-icon">
```

Here is an example of a favicon in a bookmarks panel:



There are lots of other icon types to consider these days as well. For example, you'll find this in the source code of the MDN Web Docs homepage:

```
<!-- third-generation iPad with high-resolution Retina display: -->
<link rel="apple-touch-icon-precomposed" sizes="144x144" href="https://de
<!-- iPhone with high-resolution Retina display: -->
<link rel="apple-touch-icon-precomposed" sizes="114x114" href="https://de
<!-- first- and second-generation iPad: -->
<link rel="apple-touch-icon-precomposed" sizes="72x72" href="https://deve
<!-- non-Retina iPhone, iPod Touch, and Android 2.1+ devices: -->
<link rel="apple-touch-icon-precomposed" href="https://developer.cdn.mozi
<!-- basic favicon -->
<link rel="shortcut icon" href="https://developer.cdn.mozilla.net/static/
```

The comments explain what each icon is used for — these elements cover things like providing a nice high resolution icon to use when the website is saved to an iPad's home screen.

Don't worry too much about implementing all these types of icon right now — this is a fairly advanced feature, and you won't be expected to have knowledge of this to progress through

the course. The main purpose here is to let you know what such things are, in case you come across them while browsing other websites' source code.

> **Note:** If your site uses a Content Security Policy (CSP) to enhance its security, the policy applies to the favicon. If you encounter problems with the favicon not loading, verify that the `Content-Security-Policy` header's `img-src` directive is not preventing access to it.

Applying CSS and JavaScript to HTML

Just about all websites you'll use in the modern day will employ CSS to make them look cool, and JavaScript to power interactive functionality, such as video players, maps, games, and more. These are most commonly applied to a web page using the `<link>` element and the `<script>` element, respectively.

- The `<link>` element should alway go inside the head of your document. This takes two attributes, `rel="stylesheet"`, which indicates that it is the document's stylesheet, and `href`, which contains the path to the stylesheet file:

```
<link rel="stylesheet" href="my-css-file.css">
```

- The `<script>` element should also go into the head, and should include a `src` attribute containing the path to the JavaScript you want to load, and `defer`, which basically instructs the browser to load the JavaScript at the same time as the page's HTML. This is useful as it makes sure that the HTML is all loaded before the JavaScript runs, so that you don't get errors resulting from JavaScript trying to access an HTML element that doesn't exist on the page yet. There are actually a number of ways to handle loading JavaScript on your page, but this is the most foolproof one to use for modern browsers (for others, read Script loading strategies).

```
<script src="my-js-file.js" defer></script>
```

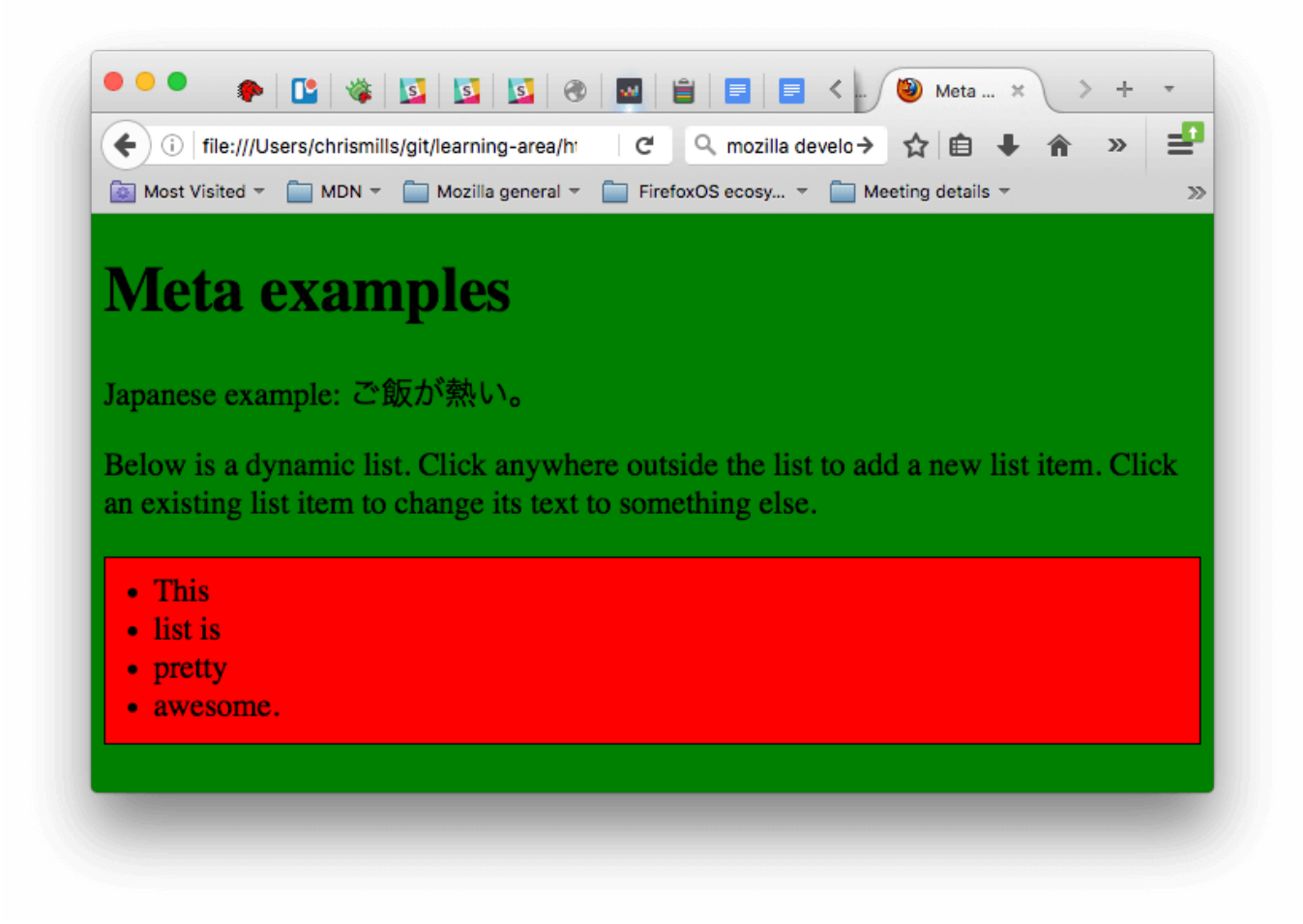
> **Note**: The `<script>` element may look like an empty element, but it's not, and so needs a closing tag. Instead of pointing to an external script file, you can also choose to

> put your script inside the `<script>` element.

Active learning: applying CSS and JavaScript to a page

1. To start this active learning, grab a copy of our meta-example.html, script.js and style.css files, and save them on your local computer in the same directory. Make sure they are saved with the correct names and file extensions.
2. Open the HTML file in both your browser, and your text editor.
3. By following the information given above, add `<link>` and `<script>` elements to your HTML, so that your CSS and JavaScript are applied to your HTML.

If done correctly, when you save your HTML and refresh your browser you should be able to see that things have changed:



- The JavaScript has added an empty list to the page. Now when you click anywhere on the list, a dialog box will pop up asking you to enter some text for a new list item. When you press the OK button, a new list item will be added to the list containing the text. When

you click on an existing list item, a dialog box will pop up allowing you to change the item's text.

- The CSS has caused the background to go green, and the text to become bigger. It has also styled some of the content that the JavaScript has added to the page (the red bar with the black border is the styling the CSS has added to the JS-generated list.)

> **Note**: If you get stuck in this exercise and can't get the CSS/JS to apply, try checking out our css-and-js.html example page.

Setting the primary language of the document

Finally, it's worth mentioning that you can (and really should) set the language of your page. This can be done by adding the lang attribute to the opening HTML tag (as seen in the meta-example.html and shown below.)

```
1 <html lang="en-US">
```

This is useful in many ways. Your HTML document will be indexed more effectively by search engines if its language is set (allowing it to appear correctly in language-specific results, for example), and it is useful to people with visual impairments using screen readers (for example, the word "six" exists in both French and English, but is pronounced differently.)

You can also set subsections of your document to be recognised as different languages. For example, we could set our Japanese language section to be recognised as Japanese, like so:

```
1 <p>Japanese example: <span lang="ja">ご飯が熱い。</span>.</p>
```

These codes are defined by the ISO 639-1 standard. You can find more about them in Language tags in HTML and XML.

Summary

That marks the end of our quickfire tour of the HTML head — there's a lot more you can do in here, but an exhaustive tour would be boring and confusing at this stage, and we just wanted to give you an idea of the most common things you'll find in there for now! In the next article we'll be looking at HTML text fundamentals.

Previous Overview: Introduction to HTML Next

In this module

Last modified: May 11, 2020, by MDN contributors

Related Topics

Complete beginners start here!

- Getting started with the Web

HTML text fundamentals

Previous Overview: Introduction to HTML Next

One of HTML's main jobs is to give text structure and meaning (also known as semantics) so that a browser can display it correctly. This article explains the way HTML can be used to structure a page of text by adding headings and paragraphs, emphasizing words, creating lists, and more.

Prerequisites:	Basic HTML familiarity, as covered in Getting started with HTML.
Objective:	Learn how to mark up a basic page of text to give it structure and meaning — including paragraphs, headings, lists, emphasis, and quotations.

The basics: Headings and Paragraphs

Most structured text consists of headings and paragraphs, whether you are reading a story, a newspaper, a college textbook, a magazine, etc.

The Tribune Prints the News

Hilo Tribune.

We Lead Others Follow

VOL. 10. HILO, HAWAII, HAWAIIAN ISLANDS, TUESDAY, FEBRUARY 28, 1905. No. 18.

The Hilo Tribune.

Drs. Grace and Irwin

ATTORNEYS-AT-LAW.

Chas. M. LeBlond

ATTORNEY-AT-LAW

Ridgway & Ridgway

ATTORNEYS-AT-LAW

Mortgagee's Notice of Intention to Foreclose.

KILAUEA CRATER CONTINUES IN ERUPTION

FOUR DAYS OF ACTIVITY
ANOTHER OUTBREAK OCCURS

LATER—Just as the TRIBUNE goes to press a telephone message from the Volcano House brings the news of renewed activity, more grand if possible than before. Parties returning from the crater report that the fire has burst through the landslide which choked the lava flow and a magnificent fountain is playing continuously, the extent of the opening in the side wall being 250 feet in length by 100 feet in width.

HONOLULU GETS APPROPRIATIONS FOR HARBOR AND LIGHTHOUSE

River and Harbor Bill Passes.

Postmaster Pratt Confirmed.

Latest Sugar Quotations.

Fire At New Orleans.

Structured content makes the reading experience easier and more enjoyable.

In HTML, each paragraph has to be wrapped in a `<p>` element, like so:

```
<p>I am a paragraph, oh yes I am.</p>
```

Each heading has to be wrapped in a heading element:

```
<h1>I am the title of the story.</h1>
```

There are six heading elements — `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`, and `<h6>`. Each element represents a different level of content in the document; `<h1>` represents the main heading, `<h2>` represents subheadings, `<h3>` represents sub-subheadings, and so on.

Implementing structural hierarchy

As an example, in a story the `<h1>` element would represent the title of the story, `<h2>` elements would represent the title of each chapter, `<h3>` elements would represent sub-sections of each chapter, and so on.

```
<h1>The Crushing Bore</h1>

<p>By Chris Mills</p>

<h2>Chapter 1: The dark night</h2>

<p>It was a dark night. Somewhere, an owl hooted. The rain lashed down or

<h2>Chapter 2: The eternal silence</h2>

<p>Our protagonist could not so much as a whisper out of the shadowy figu

<h3>The specter speaks</h3>

<p>Several more hours had passed, when all of a sudden the specter sat bo
```

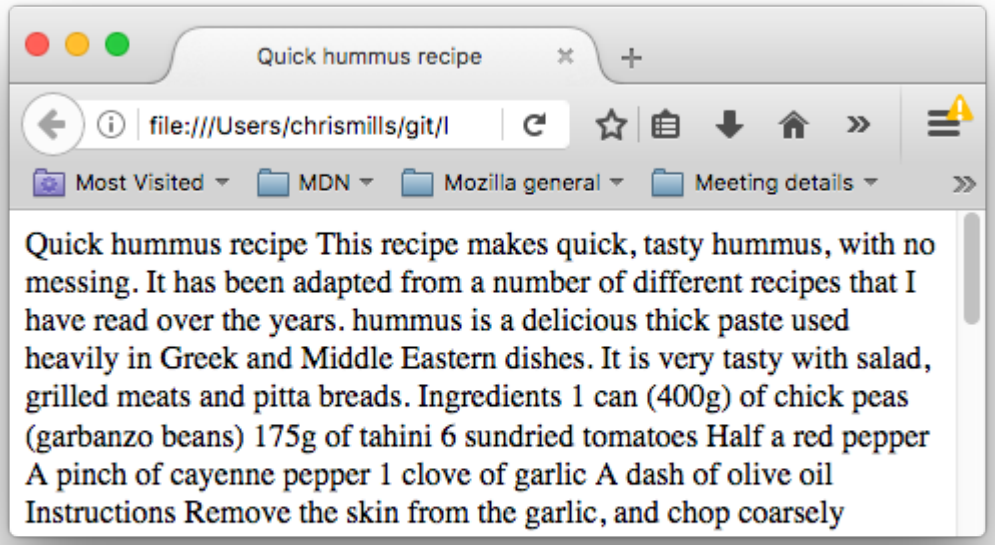
It's really up to you what exactly the elements involved represent, as long as the hierarchy makes sense. You just need to bear in mind a few best practices as you create such structures:

- Preferably you should just use a single `<h1>` per page — this is the top level heading, and all others sit below this in the hierarchy.
- Make sure you use the headings in the correct order in the hierarchy. Don't use `<h3>` elements to represent subheadings, followed by `<h2>` elements to represent sub-subheadings — that doesn't make sense and will lead to weird results.
- Of the six heading levels available, you should aim to use no more than three per page, unless you feel it is necessary. Documents with many levels (i.e., a deep heading hierarchy) become unwieldy and difficult to navigate. On such occasions, it is advisable to spread the content over multiple pages if possible.

Why do we need structure?

To answer this question, let's take a look at text-start.html — the starting point of our running example for this article (a nice hummus recipe). You should save a copy of this file on your local machine, as you'll need it for the exercises later on. This document's body currently contains multiple pieces of content — they aren't marked up in any way, but they are separated with linebreaks (Enter/Return pressed to go onto the next line).

However, when you open the document in your browser, you'll see that the text appears as a big chunk!



This is because there are no elements to give the content structure, so the browser does not know what is a heading and what is a paragraph. Furthermore:

- Users looking at a web page tend to scan quickly to find relevant content, often just reading the headings to begin with (we usually spend a very short time on a web page). If they can't see anything useful within a few seconds, they'll likely get frustrated and go somewhere else.
- Search engines indexing your page consider the contents of headings as important keywords for influencing the page's search rankings. Without headings, your page will perform poorly in terms of SEO (Search Engine Optimization).
- Severely visually impaired people often don't read web pages; they listen to them instead. This is done with software called a screen reader. This software provides ways to get fast access to given text content. Among the various techniques used, they provide an outline of the document by reading out the headings, allowing their users to find the information they need quickly. If headings are not available, they will be forced to listen to the whole document read out loud.
- To style content with CSS, or make it do interesting things with JavaScript, you need to have elements wrapping the relevant content, so CSS/JavaScript can effectively target it.

We therefore need to give our content structural markup.

Active learning: Giving our content structure

Let's jump straight in with a live example. In the example below, add elements to the raw text in the *Input* field so that it appears as a heading and two paragraphs in the *Output* field.

If you make a mistake, you can always reset it using the *Reset* button. If you get stuck, press the *Show solution* button to see the answer.

Live output

My short story I am a policewoman and my name is Trish. My legs are made of cardboard and I am married to a fish.

Editable code

Press Esc to move focus away from the code area (Tab inserts a tab character).

```
My short story I am a policewoman and my name is Trish.

My legs are made of cardboard and I am married to a fish.
```

Reset Show solution

Why do we need semantics?

Semantics are relied on everywhere around us — we rely on previous experience to tell us what the function of an everyday object is; when we see something, we know what its function will be. So, for example, we expect a red traffic light to mean "stop", and a green traffic light to mean "go". Things can get tricky very quickly if the wrong semantics are applied (Do any countries use red to mean "go"? I hope not.)

In a similar vein, we need to make sure we are using the correct elements, giving our content the correct meaning, function, or appearance. In this context the `<h1>` element is also a

semantic element, which gives the text it wraps around the role (or meaning) of "a top level heading on your page."

```
<h1>This is a top level heading</h1>
```

By default, the browser will give it a large font size to make it look like a heading (although you could style it to look like anything you wanted using CSS). More importantly, its semantic value will be used in multiple ways, for example by search engines and screen readers (as mentioned above).

On the other hand, you could make any element *look* like a top level heading. Consider the following:

```
<span style="font-size: 32px; margin: 21px 0; display: block;">Is this a
```

This is a `<span>` element. It has no semantics. You use it to wrap content when you want to apply CSS to it (or do something to it with JavaScript) without giving it any extra meaning (you'll find out more about these later on in the course). We've applied some CSS to it to make it look like a top level heading, but since it has no semantic value, it will not get any of the extra benefits described above. It is a good idea to use the relevant HTML element for the job.

Lists

Now let's turn our attention to lists. Lists are everywhere in life — from your shopping list to the list of directions you subconsciously follow to get to your house every day, to the lists of instructions you are following in these tutorials! Lists are everywhere on the Web too, and we've got three different types to worry about.

Unordered

Unordered lists are used to mark up lists of items for which the order of the items doesn't matter — let's take a shopping list as an example.

```
milk
eggs
bread
hummus
```

Every unordered list starts off with a `<ul>` element — this wraps around all the list items:

```
<ul>
milk
eggs
bread
hummus
</ul>
```

The last step is to wrap each list item in a `<li>` (list item) element:

```
<ul>
  <li>milk</li>
  <li>eggs</li>
  <li>bread</li>
  <li>hummus</li>
</ul>
```

Active learning: Marking up an unordered list

Try editing the live sample below to create your very own HTML unordered list.

Live output

milk eggs bread hummus

Editable code

Press Esc to move focus away from the code area (Tab inserts a tab character).

```
milk
eggs
bread
hummus
```

Reset Show solution

Ordered

Ordered lists are lists in which the order of the items *does* matter — let's take a set of directions as an example:

```
Drive to the end of the road
Turn right
Go straight across the first two roundabouts
Turn left at the third roundabout
The school is on your right, 300 meters up the road
```

The markup structure is the same as for unordered lists, except that you have to wrap the list items in an `<ol>` element, rather than `<ul>`:

```
<ol>
  <li>Drive to the end of the road</li>
  <li>Turn right</li>
  <li>Go straight across the first two roundabouts</li>
  <li>Turn left at the third roundabout</li>

```

```
  7     <li>The school is on your right, 300 meters up the road</li>
    </ol>
```

Active learning: Marking up an ordered list

Try editing the live sample below to create your very own HTML ordered list.

Live output

Drive to the end of the road Turn right Go straight across the first two roundabouts Turn left at the third roundabout The school is on your right, 300 meters up the road

Editable code

Press Esc to move focus away from the code area (Tab inserts a tab character).

```
Drive to the end of the road
Turn right
Go straight across the first two roundabouts
Turn left at the third roundabout
The school is on your right, 300 meters up the road
```

Reset Show solution

Active learning: Marking up our recipe page

So at this point in the article, you have all the information you need to mark up our recipe page example. You can choose to either save a local copy of our text-start.html starting file and do the work there, or do it in the editable example below. Doing it locally will probably be better, as then you'll get to save the work you are doing, whereas if you fill it in to the editable example, it will be lost the next time you open the page. Both have pros and cons.

Live output

Quick hummus recipe This recipe makes quick, tasty hummus, with no messing. It has been adapted from a number of different recipes that I have read over the years. Hummus is a delicious thick paste used heavily in Greek and Middle Eastern dishes. It is very tasty with salad, grilled meats and pitta breads. Ingredients 1 can (400g) of chick peas (garbanzo beans) 175g of tahini 6 sundried tomatoes Half a red pepper A pinch of cayenne pepper 1 clove of garlic A dash of olive oil Instructions Remove the skin from the garlic, and chop coarsely Remove all the seeds and stalk from the pepper, and chop coarsely Add all the ingredients into a food processor Process all the ingredients into a paste If you want a coarse "chunky" hummus, process it for a short time If you want a smooth hummus, process it for a longer time For a different flavour, you could try blending in a small measure of lemon and coriander, chili pepper, lime and chipotle, harissa and mint, or spinach and feta cheese. Experiment and see what works for you. Storage Refrigerate the finished hummus in a sealed container. You should be able to use it for about a week after you've made it. If it starts to become fizzy, you should definitely discard it. Hummus is suitable for freezing; you should thaw it and use it within a couple of months.

Editable code

Press Esc to move focus away from the code area (Tab inserts a tab character).

```
Quick hummus recipe

  This recipe makes quick, tasty hummus, with no messing. It has been adapted from
a number of different recipes that I have read over the years.

  Hummus is a delicious thick paste used heavily in Greek and Middle Eastern
dishes. It is very tasty with salad, grilled meats and pitta breads.

  Ingredients
```

If you get stuck, you can always press the *Show solution* button, or check out our text-complete.html example on our github repo.

Nesting lists

It is perfectly ok to nest one list inside another one. You might want to have some sub-bullets sitting below a top level bullet. Let's take the second list from our recipe example:

```
<ol>
  <li>Remove the skin from the garlic, and chop coarsely.</li>
  <li>Remove all the seeds and stalk from the pepper, and chop coarsely.<
  <li>Add all the ingredients into a food processor.</li>
  <li>Process all the ingredients into a paste.</li>
  <li>If you want a coarse "chunky" hummus, process it for a short time.<
  <li>If you want a smooth hummus, process it for a longer time.</li>
</ol>
```

Since the last two bullets are very closely related to the one before them (they read like sub-instructions or choices that fit below that bullet), it might make sense to nest them inside their own unordered list, and put that list inside the current fourth bullet. This would look like so:

```
<ol>
  <li>Remove the skin from the garlic, and chop coarsely.</li>
  <li>Remove all the seeds and stalk from the pepper, and chop coarsely.<
  <li>Add all the ingredients into a food processor.</li>
  <li>Process all the ingredients into a paste.
    <ul>
      <li>If you want a coarse "chunky" hummus, process it for a short ti
      <li>If you want a smooth hummus, process it for a longer time.</li>
    </ul>
  </li>
</ol>
```

Try going back to the previous active learning example and updating the second list like this.

Emphasis and importance

In human language, we often emphasise certain words to alter the meaning of a sentence, and we often want to mark certain words as important or different in some way. HTML provides various semantic elements to allow us to mark up textual content with such effects, and in this section, we'll look at a few of the most common ones.

Emphasis

When we want to add emphasis in spoken language, we *stress* certain words, subtly altering the meaning of what we are saying. Similarly, in written language we tend to stress words by putting them in italics. For example, the following two sentences have different meanings.

I am glad you weren't late.

I am *glad* you weren't *late*.

The first sentence sounds genuinely relieved that the person wasn't late. In contrast, the second one sounds sarcastic or passive-aggressive, expressing annoyance that the person arrived a bit late.

In HTML we use the `<em>` (emphasis) element to mark up such instances. As well as making the document more interesting to read, these are recognised by screen readers and spoken out in a different tone of voice. Browsers style this as italic by default, but you shouldn't use this tag purely to get italic styling. To do that, you'd use a `<span>` element and some CSS, or perhaps an `<i>` element (see below).

```
<p>I am <em>glad</em> you weren't <em>late</em>.</p>
```

Strong importance

To emphasize important words, we tend to stress them in spoken language and **bold** them in written language. For example:

This liquid is **highly toxic**.

I am counting on you. **Do not** be late!

In HTML we use the `<strong>` (strong importance) element to mark up such instances. As well as making the document more useful, again these are recognized by screen readers and spoken in a different tone of voice. Browsers style this as bold text by default, but you shouldn't use this tag purely to get bold styling. To do that, you'd use a `<span>` element and some CSS, or perhaps a `<b>` element (see below).

```
<p>This liquid is <strong>highly toxic</strong>.</p>

<p>I am counting on you. <strong>Do not</strong> be late!</p>
```

You can nest strong and emphasis inside one another if desired:

```
<p>This liquid is <strong>highly toxic</strong> —
if you drink it, <strong>you may <em>die</em></strong>.</p>
```

Active learning: Let's be important!

In this active learning section, we have provided an editable example. Inside it, we'd like you to try adding emphasis and strong importance to the words you think need them, just to have some practice.

Live output

Important notice

On Sunday January 9th 2010, a gang of goths were spotted stealing several garden gnomes from a shopping center in downtown Milwaukee. They were all wearing green jumpsuits and silly hats, and seemed to be having a whale of a time. If anyone has any information about this incident, please contact the police now.

Editable code

Press Esc to move focus away from the code area (Tab inserts a tab character).

```
<h1>Important notice</h1>
<p>On Sunday January 9th 2010, a gang of goths were
  spotted stealing several garden gnomes from a
  shopping center in downtown Milwaukee. They were
  all wearing green jumpsuits and silly hats, and
  seemed to be having a whale of a time. If anyone
   has any information about this incident, please
    contact the police now.</p>
```

Reset Show solution

Italic, bold, underline...

The elements we've discussed so far have clearcut associated semantics. The situation with `<b>`, `<i>`, and `<u>` is somewhat more complicated. They came about so people could write bold, italics, or underlined text in an era when CSS was still supported poorly or not at all. Elements like this, which only affect presentation and not semantics, are known as

presentational elements and should no longer be used, because as we've seen before, semantics is so important to accessibility, SEO, etc.

HTML5 redefined `<b>`, `<i>` and `<u>` with new, somewhat confusing, semantic roles.

Here's the best rule of thumb: it's likely appropriate to use `<b>`, `<i>`, or `<u>` to convey a meaning traditionally conveyed with bold, italics, or underline, provided there is no more suitable element. However, it always remains critical to keep an accessibility mindset. The concept of italics isn't very helpful to people using screen readers, or to people using a writing system other than the Latin alphabet.

- `<i>` is used to convey a meaning traditionally conveyed by italic: Foreign words, taxonomic designation, technical terms, a thought...
- `<b>` is used to convey a meaning traditionally conveyed by bold: Key words, product names, lead sentence...
- `<u>` is used to convey a meaning traditionally conveyed by underline: Proper name, misspelling...

> A kind warning about underline: **People strongly associate underlining with hyperlinks.** Therefore, on the Web, it's best to underline only links. Use the `<u>` element when it's semantically appropriate, but consider using CSS to change the default underline to something more appropriate on the Web. The example below illustrates how it can be done.

```
<!-- scientific names -->
<p>
  The Ruby-throated Hummingbird (<i>Archilochus colubris</i>)
  is the most common hummingbird in Eastern North America.
</p>

<!-- foreign words -->
<p>
  The menu was a sea of exotic words like <i lang="uk-latn">vatrushka</i>
  <i lang="id">nasi goreng</i> and <i lang="fr">soupe à l'oignon</i>.
</p>

<!-- a known misspelling -->
<p>
  Someday I'll learn how to <u style="text-decoration-line: underline; te
```

```
</p>

<!-- Highlight keywords in a set of instructions -->
<ol>
  <li>
    <b>Slice</b> two pieces of bread off the loaf.
  </li>
  <li>
    <b>Insert</b> a tomato slice and a leaf of
    lettuce between the slices of bread.
  </li>
</ol>
```

Test your skills!

You've reached the end of this article, but can you remember the most important information? You can find some further tests to verify that you've retained this information before you move on — see Test your skills: HTML text basics.

Summary

That's it for now! This article should have given you a good idea of how to start marking up text in HTML, and introduced you to some of the most important elements in this area. There are a lot more semantic elements to cover in this area, and we'll look at a lot more in our Advanced text formatting article, later on in the course. In the next article, we'll be looking in detail at how to create hyperlinks, possibly the most important element on the Web.

Previous Overview: Introduction to HTML Next

Creating hyperlinks

Previous Overview: Introduction to HTML Next

Hyperlinks are really important — they are what makes the Web *a web*. This article shows the syntax required to make a link, and discusses link best practices.

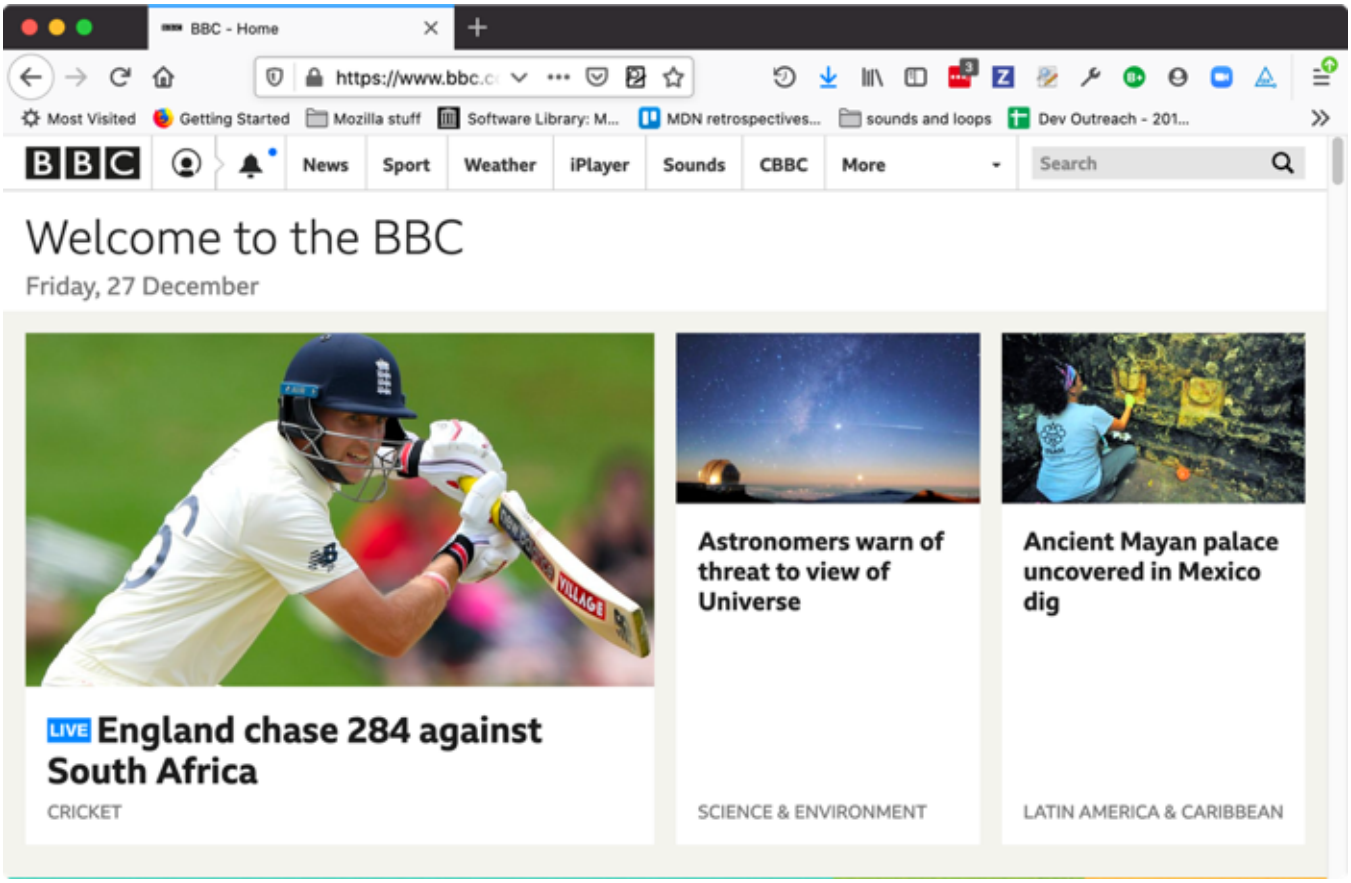
Prerequisites:	Basic HTML familiarity, as covered in Getting started with HTML. HTML text formatting, as covered in HTML text fundamentals.
Objective:	To learn how to implement a hyperlink effectively, and link multiple files together.

What is a hyperlink?

Hyperlinks are one of the most exciting innovations the Web has to offer. Well, they've been a feature of the Web since the very beginning, but they are what makes the Web *a Web* — they allow us to link our documents to any other document (or other resource) we want to, we can also link to specific parts of documents, and we can make apps available at a simple web address (contrast this to native apps, which have to be installed and all that business.) Just about any web content can be converted to a link, so that when clicked (or otherwise activated) it will make the web browser go to another web address (URL).

> **Note**: A URL can point to HTML files, text files, images, text documents, video and audio files, and anything else that can live on the Web. If the web browser doesn't know how to display or handle the file, it will ask you if you want to open the file (in which case the duty of opening or handling the file is passed to a suitable native app on the device) or download the file (in which case you can try to deal with it later on.)

The BBC homepage, for example, contains a large number of links that point not only to multiple news stories, but also different areas of the site (navigation functionality), login/registration pages (user tools) and more.



Anatomy of a link

A basic link is created by wrapping the text (or other content, see Block level links) you want to turn into a link inside an `<a>` element, and giving it an `href` attribute (also known as a **Hypertext Reference** , or **target**) that will contain the web address you want the link to point to.

```
<p>I'm creating a link to
<a href="https://www.mozilla.org/en-US/">the Mozilla homepage</a>.
</p>
```

This gives us the following result:

I'm creating a link to the Mozilla homepage.

Adding supporting information with the title attribute

Another attribute you may want to add to your links is `title`; this is intended to contain supplementary useful information about the link, such as what kind of information the page contains, or things to be aware of. For example:

```
<p>I'm creating a link to
<a href="https://www.mozilla.org/en-US/"
   title="The best place to find more information about Mozilla's
          mission and how to contribute">the Mozilla homepage</a>.
</p>
```

This gives us the following result (the title will come up as a tooltip when the link is hovered over):

I'm creating a link to the Mozilla homepage.

> **Note**: A link title is only revealed on mouse hover, which means that people relying on keyboard controls or touchscreens to navigate web pages will have difficulty accessing title information. If a title's information is truly important to the usability of page, then you should present it in a manner that will be accessible to all users, for example by putting it in the regular text.

Active learning: creating your own example link

Active learning time: we'd like you to create an HTML document using your local code editor (our getting started template would do just fine.)

- Inside the HTML body, try adding one or more paragraphs or other types of content you already know about.
- Turn some of the content into links.
- Include title attributes.

Block level links

As mentioned before, you can turn just about any content into a link, even block level elements. If you had an image you wanted to turn into a link, you could just put the image between `<a>` `</a>` tags.

```
<a href="https://www.mozilla.org/en-US/">
  <img src="mozilla-image.png" alt="mozilla logo that links to the mozil
</a>
```

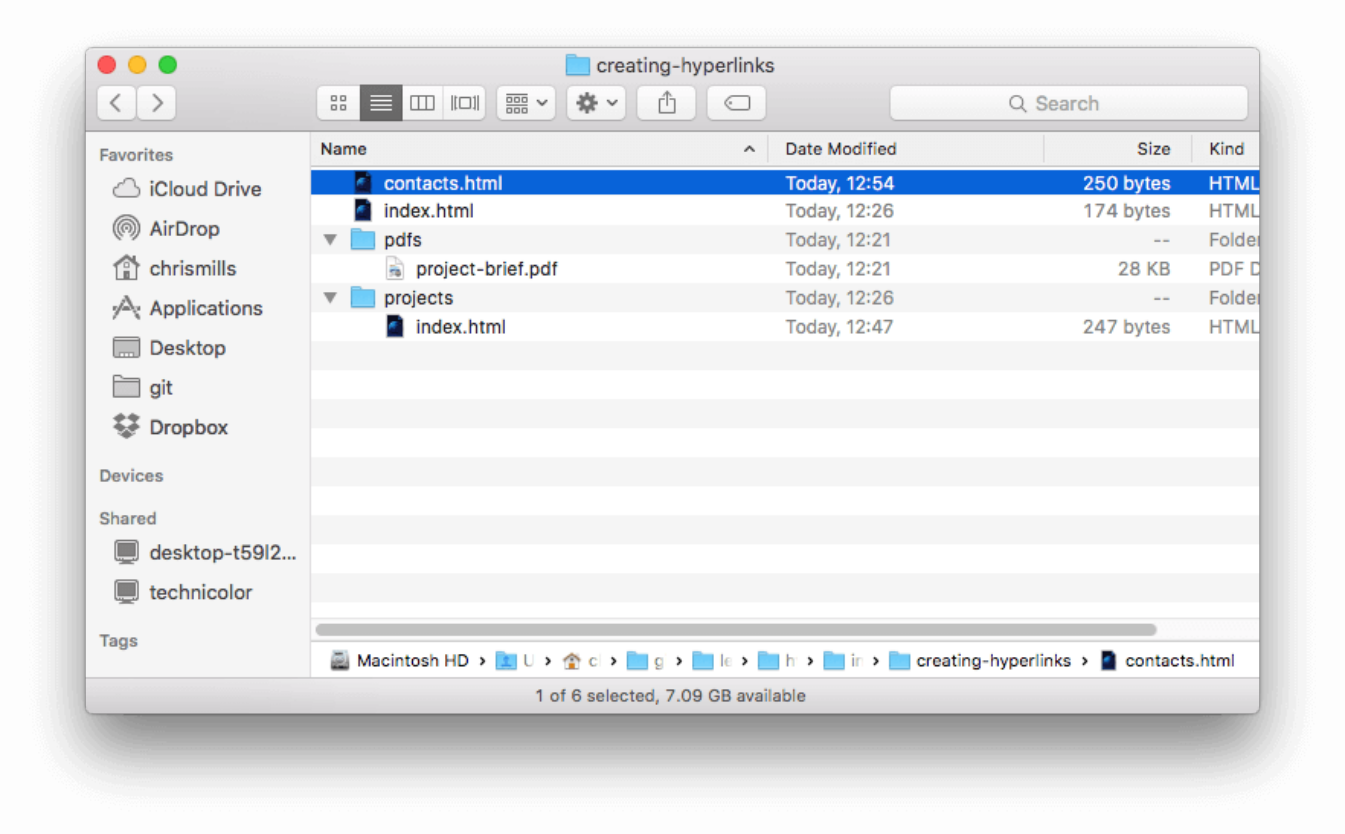
Note: You'll find out a lot more about using images on the Web in a future article.

A quick primer on URLs and paths

To fully understand link targets, you need to understand URLs and file paths. This section gives you the information you need to achieve this.

A URL, or Uniform Resource Locator is simply a string of text that defines where something is located on the Web. For example, Mozilla's English homepage is located at `https://www.mozilla.org/en-US/`.

URLs use paths to find files. Paths specify where in the filesystem the file you are interested in is located. Let's look at a simple example of a directory structure (see the creating-hyperlinks directory.)



The **root** of this directory structure is called `creating-hyperlinks`. When working locally with a web site, you will have one directory that the whole site goes inside. Inside the root, we have an `index.html` file and a `contacts.html`. In a real website, `index.html` would be our home page or landing page (a web page that serves as the entry point for a website or a particular section of a website.).

There are also two directories inside our root — `pdfs` and `projects`. These each have a single file inside them — a PDF (`project-brief.pdf`) and an `index.html` file, respectively. Note that you can quite happily have two `index.html` files in one project, as long as they are in different locations in the filesystem. Many web sites do. The second `index.html` would perhaps be the main landing page for project-related information.

- **Same directory**: If you wanted to include a hyperlink inside `index.html` (the top level `index.html`) pointing to `contacts.html`, you would just need to specify the filename of the file you want to link to, as it is in the same directory as the current file. So the URL you would use is `contacts.html`:

```
<p>Want to contact a specific staff member?
Find details on our <a href="contacts.html">contacts page</a>.</p
```

- **Moving down into subdirectories**: If you wanted to include a hyperlink inside `index.html` (the top level `index.html`) pointing to `projects/index.html`, you would need to go down into the `projects` directory before indicating the file you want to link to. This is done by specifying the directory's name, then a forward slash, then the name of the file. so the URL you would use is `projects/index.html`:

```
<p>Visit my <a href="projects/index.html">project homepage</a>.</
```

- **Moving back up into parent directories**: If you wanted to include a hyperlink inside `projects/index.html` pointing to `pdfs/project-brief.pdf`, you'd have to go up a directory level, then back down into the `pdf` directory. "Go up a directory" is indicated using two dots — `..` — so the URL you would use is `../pdfs/project-brief.pdf`:

```
<p>A link to my <a href="../pdfs/project-brief.pdf">project brief
```

> **Note**: You can combine multiple instances of these features into complex URLs, if needed, e.g. `../../../complex/path/to/my/file.html`.

Document fragments

It is possible to link to a specific part of an HTML document (known as a **document fragment**), rather than just to the top of the document. To do this you first have to assign an `id` attribute to the element you want to link to. It normally makes sense to link to a specific heading, so this would look something like the following:

```
<h2 id="Mailing_address">Mailing address</h2>
```

Then to link to that specific `id`, you'd include it at the end of the URL, preceded by a hash/pound symbol, for example:

```
<p>Want to write us a letter? Use our <a href="contacts.html#Mailing_addr
```

You can even use the document fragment reference on its own to link to *another part of the same document*:

```
<p>The <a href="#Mailing_address">company mailing address</a> can be four
```

Absolute versus relative URLs

Two terms you'll come across on the Web are **absolute URL** and **relative URL:**

absolute URL: Points to a location defined by its absolute location on the web, including protocol and domain name. So for example, if an `index.html` page is uploaded to a directory called `projects` that sits inside the root of a web server, and the web site's domain is `http://www.example.com`, the page would be available at `http://www.example.com/projects/index.html` (or even just `http://www.example.com/projects/`, as most web servers just look for a landing page such as `index.html` to load if it is not specified in the URL.)

An absolute URL will always point to the same location, no matter where it is used.

relative URL: Points to a location that is *relative* to the file you are linking from, more like what we looked at in the previous section. For example, if we wanted to link from our example file at `http://www.example.com/projects/index.html` to a PDF file in the same directory, the URL would just be the filename — e.g. `project-brief.pdf` — no extra information needed. If the PDF was available in a subdirectory inside `projects` called `pdfs`, the relative link would be `pdfs/project-brief.pdf` (the equivalent absolute URL would be `http://www.example.com/projects/pdfs/project-brief.pdf`.)

A relative URL will point to different places depending on the actual location of the file you refer from — for example if we moved our `index.html` file out of the `projects` directory and into the root of the web site (the top level, not in any directories), the `pdfs/project-brief.pdf` relative URL link inside it would now point to a file located at `http://www.example.com/pdfs/project-brief.pdf`, not a file located at `http://www.example.com/projects/pdfs/project-brief.pdf`.

Of course, the location of the `project-brief.pdf` file and `pdfs` folder won't suddenly change because you moved the `index.html` file — this would make your link point to the wrong place, so it wouldn't work if clicked on. You need to be careful!

Link best practices

There are some best practices to follow when writing links. Let's look at these now.

Use clear link wording

It's easy to throw links up on your page. That's not enough. We need to make our links *accessible* to all readers, regardless of their current context and which tools they prefer. For example:

- Screenreader users like jumping around from link to link on the page, and reading links out of context.
- Search engines use link text to index target files, so it is a good idea to include keywords in your link text to effectively describe what is being linked to.
- Visual readers skim over the page rather than reading every word, and their eyes will be drawn to page features that stand out, like links. They will find descriptive link text useful.

Let's look at a specific example:

***Good** link text:* Download Firefox

```
<p><a href="https://firefox.com/">
  Download Firefox
</a></p>
```

***Bad** link text:* Click here to download Firefox

```
<p><a href="https://firefox.com/">
  Click here
</a>
to download Firefox</p>
```

Other tips:

- Don't repeat the URL as part of the link text — URLs look ugly, and sound even uglier when a screen reader reads them out letter by letter.
- Don't say "link" or "links to" in the link text — it's just noise. Screen readers tell people there's a link. Visual users will also know there's a link, because links are generally styled in a different colour and underlined (this convention generally shouldn't be broken, as users are so used to it.)
- Keep your link label as short as possible — long links especially annoy screen reader users, who have to hear the whole thing read out.
- Minimize instances where multiple copies of the same text are linked to different places. This can cause problems for screenreader users, who will often bring up a list of the links out of context — several links all labelled "click here", "click here", "click here" would be confusing.

Use relative links wherever possible

From the description above, you might think that it is a good idea to just use absolute links all the time; after all, they don't break when a page is moved like relative links. However, you should use relative links wherever possible when linking to other locations within the *same website* (when linking to *another website*, you will need to use an absolute link):

- For a start, it is a lot easier to scan your code — relative URLs are generally a lot shorter than absolute URLs, which makes reading code much easier.
- Second, it is more efficient to use relative URLs wherever possible. When you use an absolute URL, the browser starts by looking up the real location of the server on the Domain Name System (DNS; see How the web works for more information), then it goes to that server and finds the file that is being requested. With a relative URL on the other hand, the browser just looks up the file that is being requested, on the same server. So if you use absolute URLs where relative URLs would do, you are constantly making your browser do extra work, meaning that it will perform less efficiently.

Linking to non-HTML resources — leave clear signposts

When linking to a resource that will be downloaded (like a PDF or Word document) or streamed (like video or audio) or has another potentially unexpected effect (opens a popup window, or loads a Flash movie), you should add clear wording to reduce any confusion. It can be quite annoying for example:

- If you are on a low bandwidth connection, click a link and then a multiple megabyte download starts unexpectedly.
- If you haven't got the Flash player installed, click a link and then suddenly get taken to a page that requires Flash.

Let's look at some examples, to see what kind of text can be used here:

```
<p><a href="http://www.example.com/large-report.pdf">
  Download the sales report (PDF, 10MB)
</a></p>

<p><a href="http://www.example.com/video-stream/" target="_blank">
  Watch the video (stream opens in separate tab, HD quality)
</a></p>

<p><a href="http://www.example.com/car-game">
  Play the car game (requires Flash)
</a></p>
```

Use the download attribute when linking to a download

When you are linking to a resource that is to be downloaded rather than opened in the browser, you can use the `download` attribute to provide a default save filename. Here's an example with a download link to the latest Windows version of Firefox:

```
<a href="https://download.mozilla.org/?product=firefox-latest-ssl&os=win6
   download="firefox-latest-64bit-installer.exe">
  Download Latest Firefox for Windows (64-bit) (English, US)
</a>
```

Active learning: creating a navigation menu

For this exercise, we'd like you to link some pages together with a navigation menu to create a multi-page website. This is one common way in which a website is created — the same page structure is used on every page, including the same navigation menu, so when links are clicked it gives the impression that you are staying in the same place, and different content is being brought up.

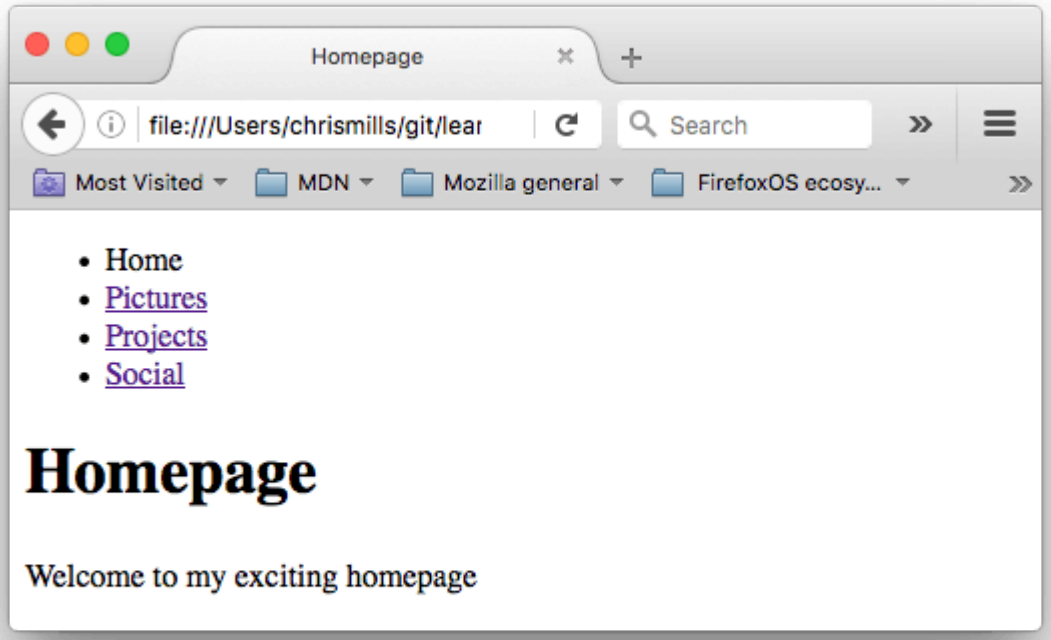
You'll need to make local copies of the following four pages, all in the same directory (see also the navigation-menu-start directory for a full file listing):

- index.html
- projects.html
- pictures.html
- social.html

You should:

1. Add an unordered list in the indicated place on one page, containing the names of the pages to link to. A navigation menu is usually just a list of links, so this is semantically ok.
2. Turn each page name into a link to that page.
3. Copy the navigation menu across to each page.
4. On each page, remove just the link to that same page — it is confusing and pointless for a page to include a link to itself, and the lack of a link acts a good visual reminder of what page you are currently on.

The finished example should end up looking something like this:



> **Note**: If you get stuck, or are not sure if you have got it right, you can check the navigation-menu-marked-up directory to see the correct answer.

E-mail links

It is possible to create links or buttons that, when clicked, open a new outgoing email message rather than linking to a resource or page. This is done using the `<a>` element and the `mailto:` URL scheme.

In its most basic and commonly used form, a `mailto:` link simply indicates the email address of the intended recipient. For example:

```
<a href="mailto:nowhere@mozilla.org">Send email to nowhere</a>
```

This results in a link that looks like this: Send email to nowhere.

In fact, the email address is even optional. If you leave it out (that is, your `href` is simply "mailto:"), a new outgoing email window will be opened by the user's mail client that has no destination address specified yet. This is often useful as "Share" links that users can click to send an email to an address of their choosing.

Specifying details

In addition to the email address, you can provide other information. In fact, any standard mail header fields can be added to the `mailto` URL you provide. The most commonly used of these are "subject", "cc", and "body" (which is not a true header field, but allows you to specify a short content message for the new email). Each field and its value is specified as a query term.

Here's an example that includes a cc, bcc, subject and body:

```
<a href="mailto:nowhere@mozilla.org?cc=name2@rapidtables.com&bcc=name3@ra
  Send mail with cc, bcc, subject and body
</a>
```

> **Note:** The values of each field must be URL-encoded, that is with non-printing characters (invisible characters like tabs, carriage returns, and page breaks) and spaces percent-escaped. Also note the use of the question mark (`?`) to separate the main URL from the field values, and ampersands (&) to separate each field in the `mailto:` URL. This is standard URL query notation. Read The GET method to understand what URL query notation is more comonly used for.

Here are a few other sample `mailto` URLs:

- mailto:
- mailto:nowhere@mozilla.org
- mailto:nowhere@mozilla.org,nobody@mozilla.org
- mailto:nowhere@mozilla.org?cc=nobody@mozilla.org
- mailto:nowhere@mozilla.org?cc=nobody@mozilla.org&subject=This%20is%20the%20subject

Test your skills!

You've reached the end of this article, but can you remember the most important information? You can find some further tests to verify that you've retained this information before you move on — see Test your skills: Links.

Summary

That's it for links, for now anyway! You'll return to links later on in the course when you start to look at styling them. Next up for HTML, we'll return to text semantics and look at some more advanced/unusual features that you'll find useful — Advanced text formatting is your next stop.

Previous Overview: Introduction to HTML Next

In this module

Advanced text formatting

Previous Overview: Introduction to HTML Next

There are many other elements in HTML for formatting text, which we didn't get to in the HTML text fundamentals article. The elements described in this article are less known, but still useful to know about (and this is still not a complete list by any means). Here you'll learn about marking up quotations, description lists, computer code and other related text, subscript and superscript, contact information, and more.

Prerequisites:	Basic HTML familiarity, as covered in Getting started with HTML. HTML text formatting, as covered in HTML text fundamentals.
Objective:	To learn how to use lesser-known HTML elements to mark up advanced semantic features.

Description lists

In HTML text fundamentals, we walked through how to mark up basic lists in HTML, but we didn't mention the third type of list you'll occasionally come across — **description lists**. The purpose of these lists is to mark up a set of items and their associated descriptions, such as terms and definitions, or questions and answers. Let's look at an example of a set of terms and definitions:

```
soliloquy
In drama, where a character speaks to themselves, representing their inne
monologue
In drama, where a character speaks their thoughts out loud to share them
aside
In drama, where a character shares a comment only with the audience for h
```

Description lists use a different wrapper than the other list types — `<dl>`; in addition each term is wrapped in a `<dt>` (description term) element, and each description is wrapped in a `<dd>` (description definition) element. Let's finish marking up our example:

```
<dl>
  <dt>soliloquy</dt>
  <dd>In drama, where a character speaks to themselves, representing thei
  <dt>monologue</dt>
  <dd>In drama, where a character speaks their thoughts out loud to share
  <dt>aside</dt>
  <dd>In drama, where a character shares a comment only with the audience
</dl>
```

The browser default styles will display description lists with the descriptions indented somewhat from the terms.

soliloquy
: In drama, where a character speaks to themselves, representing their inner thoughts or feelings and in the process relaying them to the audience (but not to other characters.)

monologue
: In drama, where a character speaks their thoughts out loud to share them with the audience and any other characters present.

aside
: In drama, where a character shares a comment only with the audience for humorous or dramatic effect. This is usually a feeling, thought, or piece of additional background information.

Note that it is permitted to have a single term with multiple descriptions, for example:

```
<dl>
  <dt>aside</dt>
  <dd>In drama, where a character shares a comment only with the audience
  <dd>In writing, a section of content that is related to the current top
</dl>
```

aside

In drama, where a character shares a comment only with the audience for humorous or dramatic effect. This is usually a feeling, thought, or piece of additional background information.
In writing, a section of content that is related to the current topic, but doesn't fit directly into the main flow of content so is presented nearby (often in a box off to the side.)

Active learning: Marking up a set of definitions

It's time to try your hand at description lists; add elements to the raw text in the *Input* field so that it appears as a description list in the *Output* field. You could try using your own terms and descriptions if you like.

If you make a mistake, you can always reset it using the *Reset* button. If you get really stuck, press the *Show solution* button to see the answer.

Live output

Bacon The glue that binds the world together. Eggs The glue that binds the cake together. Coffee The drink that gets the world running in the morning. A light brown color.

Editable code

Press Esc to move focus away from the code area (Tab inserts a tab character).

```
Bacon
The glue that binds the world together.
Eggs
The glue that binds the cake together.
Coffee
The drink that gets the world running in the morning.
A light brown color.
```

Reset Show solution

Quotations

HTML also has features available for marking up quotations; which element you use depends on whether you are marking up a block or inline quotation.

Blockquotes

If a section of block level content (be it a paragraph, multiple paragraphs, a list, etc.) is quoted from somewhere else, you should wrap it inside a `<blockquote>` element to signify this, and include a URL pointing to the source of the quote inside a `cite` attribute. For example, the following markup is taken from the MDN `<blockquote>` element page:

```
<p>The <strong>HTML <code>&lt;blockquote&gt;</code> Element</strong> (or
Quotation Element</em>) indicates that the enclosed text is an extended c
```

To turn this into a block quote, we would just do this:

```
<p>Here below is a blockquote..</p>
<blockquote cite="https://developer.mozilla.org/en-US/docs/Web/HTML/Eleme
  <p>The <strong>HTML <code>&lt;blockquote&gt;</code> Element</strong> (o
  Quotation Element</em>) indicates that the enclosed text is an extendec
</blockquote>
```

Browser default styling will render this as an indented paragraph, as an indicator that it is a quote; the paragraph above the quotation is there to demostrate that.

Here below is a blockquote..

> The **HTML `<blockquote>` Element** (or *HTML Block Quotation Element*) indicates that the enclosed text is an extended quotation.

Inline quotations

Inline quotations work in exactly the same way, except that they use the `<q>` element. For example, the below bit of markup contains a quotation from the MDN `<q>` page:

```
<p>The quote element — <code>&lt;q&gt;</code> — is <q cite="https://devel
for short quotations that don't require paragraph breaks.</q></p>
```

Browser default styling will render this as normal text put in quotes to indicate a quotation, like so:

The quote element — `<q>` — is "intended for short quotations that don't require paragraph breaks."

Citations

The content of the `cite` attribute sounds useful, but unfortunately browsers, screenreaders, etc. don't really do much with it. There is no way to get the browser to display the contents of

`cite`, without writing your own solution using JavaScript or CSS. If you want to make the source of the quotation available on the page you need to make it available in the text via a link or some other appropriate way.

There is a `<cite>` element, but this is meant to contain the title of the resource being quoted, e.g. the name of the book. There is no reason however why you couldn't link the text inside `<cite>` to the quote source in some way:

```
<p>According to the <a href="https://developer.mozilla.org/en-US/docs/Web
<cite>MDN blockquote page</cite></a>:
</p>

<blockquote cite="https://developer.mozilla.org/en-US/docs/Web/HTML/Eleme
  <p>The <strong>HTML <code>&lt;blockquote&gt;</code> Element</strong> (o
  Quotation Element</em>) indicates that the enclosed text is an extended
</blockquote>

<p>The quote element — <code>&lt;q&gt;</code> — is <q cite="https://devel
for short quotations that don't require paragraph breaks.</q> -- <a href=
<cite>MDN q page</cite></a>.</p>
```

Citations are styled in italic font by default.

According to the *MDN blockquote page*:

> The **HTML `<blockquote>` Element** (or *HTML Block Quotation Element*) indicates that the enclosed text is an extended quotation.

The quote element — `<q>` — is "intended for short quotations that don't require paragraph breaks." -- *MDN q page*.

Active learning: Who said that?

Time for another active learning example! In this example we'd like you to:

1. Turn the middle paragraph into a blockquote, which includes a `cite` attribute.

2. Turn part of the third paragraph into an inline quote, which includes a `cite` attribute.
3. Include a `<cite>` element for each link to say what the title of the quoted source is.

The citation sources you need are:

- http://www.brainyquote.com/quotes/authors/c/confucius.html for the Confucius quote
- http://www.affirmationsforpositivethinking.com/index.htm for "The Need To Eliminate Negative Self Talk".

If you make a mistake, you can always reset it using the *Reset* button. If you get really stuck, press the *Show solution* button to see the answer.

Live output

Hello and welcome to my motivation page. As Confucius' quotes site says:

It does not matter how slowly you go as long as you do not stop.

I also love the concept of positive thinking, and The Need To Eliminate Negative Self Talk (as mentioned in Affirmations for Positive Thinking.)

Editable code

Press Esc to move focus away from the code area (Tab inserts a tab character).

```
<p>Hello and welcome to my motivation page. As Confucius'
quotes site says:</p>
<p>It does not matter how slowly you go as long as you do not
stop.</p>
<p>I also love the concept of positive thinking, and The Need
To Eliminate Negative Self Talk (as mentioned in Affirmations
for Positive Thinking.)</p>
```

Reset Show solution

Abbreviations

Another fairly common element you'll meet when looking around the Web is `<abbr>` — this is used to wrap around an abbreviation or acronym, and provide a full expansion of the term

(included inside a `title` attribute.) Let's look at a couple of examples:

```
<p>We use <abbr title="Hypertext Markup Language">HTML</abbr> to structur

<p>I think <abbr title="Reverend">Rev.</abbr> Green did it in the kitcher
```

These will come out looking something like this (the expansion will appear in a tooltip when the term is hovered over):

We use HTML to structure our web documents.

I think Rev. Green did it in the kitchen with the chainsaw.

> **Note**: There is another element, `<acronym>`, which basically does the same thing as `<abbr>`, and was intended specifically for acronyms rather than abbreviations. This however has fallen into disuse — it wasn't supported in browsers as well as `<abbr>`, and has such a similar function that it was considered pointless to have both. Just use `<abbr>` instead.

Active learning: marking up an abbreviation

For this simple active learning assignment, we'd like you to simply mark up an abbreviation. You can use our sample below, or replace it with one of your own.

Live output

NASA sure does some exciting work.

Editable code

Press Esc to move focus away from the code area (Tab inserts a tab character).

```
<p>NASA sure does some exciting work.</p>
```

Reset Show solution

Marking up contact details

HTML has an element for marking up contact details — `<address>`. This simply wraps around your contact details, for example:

```
<address>
  <p>Chris Mills, Manchester, The Grim North, UK</p>
</address>
```

It could also include more complex markup, and other forms of contact information, for example:

```
<address>
  <p>
    Chris Mills<br>
    Manchester<br>
    The Grim North<br>
    UK
  </p>

  <ul>
```

```
    <li>Tel: 01234 567 890</li>
    <li>Email: me@grim-north.co.uk</li>
  </ul>
</address>
```

Note that something like this would also be ok, if the linked page contained the contact information:

```
<address>
  <p>Page written by <a href="../authors/chris-mills/">Chris Mills</a>.</
</address>
```

Superscript and subscript

You will occasionally need to use superscript and subscript when marking up items like dates, chemical formulae, and mathematical equations so they have the correct meaning. The `<sup>` and `<sub>` elements handle this job. For example:

```
<p>My birthday is on the 25<sup>th</sup> of May 2001.</p>
<p>Caffeine's chemical formula is C<sub>8</sub>H<sub>10</sub>N<sub>4</sub></sub
<p>If x<sup>2</sup> is 9, x must equal 3 or -3.</p>
```

The output of this code looks like so:

My birthday is on the 25^{th} of May 2001.

Caffeine's chemical formula is $C_8H_{10}N_4O_2$.

If x^2 is 9, x must equal 3 or -3.

Representing computer code

There are a number of elements available for marking up computer code using HTML:

- `<code>`: For marking up generic pieces of computer code.
- `<pre>`: For retaining whitespace (generally code blocks) — if you use indentation or excess whitespace inside your text, browsers will ignore it and you will not see it on your rendered page. If you wrap the text in `<pre></pre>` tags however, your whitespace will be rendered identically to how you see it in your text editor.
- `<var>`: For specifically marking up variable names.
- `<kbd>`: For marking up keyboard (and other types of) input entered into the computer.
- `<samp>`: For marking up the output of a computer program.

Let's look at a few examples. You should try having a play with these (try grabbing a copy of our other-semantics.html sample file):

```
<pre><code>var para = document.querySelector('p');

para.onclick = function() {
  alert('Owww, stop poking me!');
}</code></pre>

<p>You shouldn't use presentational elements like <code>&lt;font&gt;</coc

<p>In the above JavaScript example, <var>para</var> represents a paragrap


<p>Select all the text with <kbd>Ctrl</kbd>/<kbd>Cmd</kbd> + <kbd>A</kbd>

<pre>$ <kbd>ping mozilla.org</kbd>
<samp>PING mozilla.org (63.245.215.20): 56 data bytes
64 bytes from 63.245.215.20: icmp_seq=0 ttl=40 time=158.233 ms</samp></pr
```

The above code will look like so:

```
var para = document.querySelector('p');

para.onclick = function() {
  alert('Owww, stop poking me!');
}
```

You shouldn't use presentational elements like `<font>` and `<center>`.

In the above JavaScript example, *para* represents a paragraph element.

Select all the text with `Ctrl`/`Cmd` + `A`.

```
$ ping mozilla.org
PING mozilla.org (63.245.215.20): 56 data bytes
64 bytes from 63.245.215.20: icmp_seq=0 ttl=40 time=158.233 ms
```

Marking up times and dates

HTML also provides the `<time>` element for marking up times and dates in a machine-readable format. For example:

```
<time datetime="2016-01-20">20 January 2016</time>
```

Why is this useful? Well, there are many different ways that humans write down dates. The above date could be written as:

- 20 January 2016
- 20th January 2016
- Jan 20 2016
- 20/01/16
- 01/20/16
- The 20th of next month
- 20e Janvier 2016
- 2016年1月20日
- And so on

But these different forms cannot be easily recognised by computers — what if you wanted to automatically grab the dates of all events in a page and insert them into a calendar? The `<time>` element allows you to attach an unambiguous, machine-readable time/date for this purpose.

The basic example above just provides a simple machine readable date, but there are many other options that are possible, for example:

```
<!-- Standard simple date -->
<time datetime="2016-01-20">20 January 2016</time>
<!-- Just year and month -->
<time datetime="2016-01">January 2016</time>
<!-- Just month and day -->
<time datetime="01-20">20 January</time>
<!-- Just time, hours and minutes -->
<time datetime="19:30">19:30</time>
<!-- You can do seconds and milliseconds too! -->
<time datetime="19:30:01.856">19:30:01.856</time>
<!-- Date and time -->
<time datetime="2016-01-20T19:30">7.30pm, 20 January 2016</time>
<!-- Date and time with timezone offset -->
<time datetime="2016-01-20T19:30+01:00">7.30pm, 20 January 2016 is 8.30pm
<!-- Calling out a specific week number -->
<time datetime="2016-W04">The fourth week of 2016</time>
```

Test your skills!

You've reached the end of this article, but can you remember the most important information? You can find some further tests to verify that you've retained this information before you move on — see Test your skills: Advanced HTML text.

Summary

That marks the end of our study of HTML text semantics. Bear in mind that what you have seen during this course is not an exhaustive list of HTML text elements — we wanted to try to cover the essentials, and some of the more common ones you will see in the wild, or at least might find interesting. To find way more HTML elements, you can take a look at our HTML element reference (the Inline text semantics section would be a great place to start.) In the next article we will look at the HTML elements you'd use to structure the different parts of an HTML document.

Previous Overview: Introduction to HTML Next

In this module

Last modified: Apr 22, 2020, by MDN contributors

Related Topics

Document and website structure

Previous Overview: Introduction to HTML Next

In addition to defining individual parts of your page (such as "a paragraph" or "an image"), HTML also boasts a number of block level elements used to define areas of your website (such as "the header", "the navigation menu", "the main content column"). This article looks into how to plan a basic website structure, and write the HTML to represent this structure.

Prerequisites:	Basic HTML familiarity, as covered in Getting started with HTML. HTML text formatting, as covered in HTML text fundamentals. How hyperlinks work, as covered in Creating hyperlinks.
Objective:	Learn how to structure your document using semantic tags, and how to work out the structure of a simple website.

Basic sections of a document

Webpages can and will look pretty different from one another, but they all tend to share similar standard components, unless the page is displaying a fullscreen video or game, is part of some kind of art project, or is just badly structured:

header:
Usually a big strip across the top with a big heading, logo, and perhaps a tagline. This usually stays the same from one webpage to another.

navigation bar:

Links to the site's main sections; usually represented by menu buttons, links, or tabs. Like the header, this content usually remains consistent from one webpage to another — having inconsistent navigation on your website will just lead to confused, frustrated users. Many web designers consider the navigation bar to be part of the header rather than an individual component, but that's not a requirement; in fact, some also argue that having the two separate is better for accessibility, as screen readers can read the two features better if they are separate.

main content:

A big area in the center that contains most of the unique content of a given webpage, for example, the video you want to watch, or the main story you're reading, or the map you want to view, or the news headlines, etc. This is the one part of the website that definitely will vary from page to page!

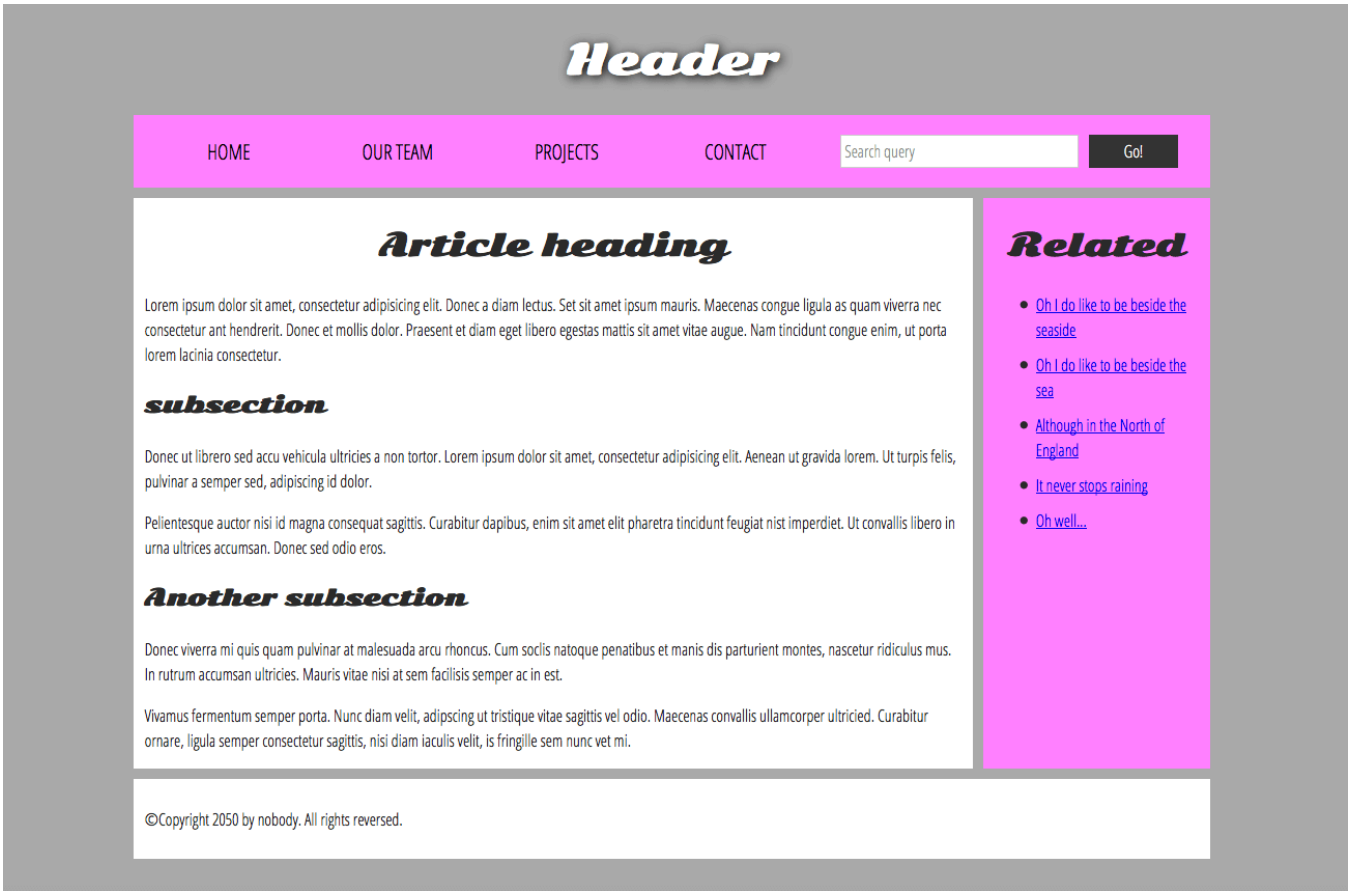
sidebar:

Some peripheral info, links, quotes, ads, etc. Usually, this is contextual to what is contained in the main content (for example on a news article page, the sidebar might contain the author's bio, or links to related articles) but there are also cases where you'll find some recurring elements like a secondary navigation system.

footer:

A strip across the bottom of the page that generally contains fine print, copyright notices, or contact info. It's a place to put common information (like the header) but usually, that information is not critical or secondary to the website itself. The footer is also sometimes used for SEO purposes, by providing links for quick access to popular content.

A "typical website" could be structured something like this:



HTML for structuring content

The simple example shown above isn't pretty, but it is perfectly fine for illustrating a typical website layout example. Some websites have more columns, some are a lot more complex, but you get the idea. With the right CSS, you could use pretty much any elements to wrap around the different sections and get it looking how you wanted, but as discussed before, we need to respect semantics and **use the right element for the right job**.

This is because visuals don't tell the whole story. We use color and font size to draw sighted users' attention to the most useful parts of the content, like the navigation menu and related links, but what about visually impaired people for example, who might not find concepts like "pink" and "large font" very useful?

> **Note**: Colorblind people represent around 4% of the world population or, to put it another way, approximately 1 in every 12 men and 1 in every 200 women are colorblind. Blind and visually impaired people represent roughly 4-5% of the world population (in 2012 there were 285 million such people in the world, while the total population was around 7 billion).

In your HTML code, you can mark up sections of content based on their *functionality* — you can use elements that represent the sections of content described above unambiguously, and assistive technologies like screenreaders can recognise those elements and help with tasks like "find the main navigation", or "find the main content." As we mentioned earlier in the course, there are a number of consequences of not using the right element structure and semantics for the right job.

To implement such semantic mark up, HTML provides dedicated tags that you can use to represent such sections, for example:

- **header:** `<header>`.
- **navigation bar:** `<nav>`.
- **main content:** `<main>`, with various content subsections represented by `<article>`, `<section>`, and `<div>` elements.
- **sidebar:** `<aside>`; often placed inside `<main>`.
- **footer:** `<footer>`.

Active learning: exploring the code for our example

Our example seen above is represented by the following code (you can also find the example in our GitHub repository). We'd like you to look at the example above, and then look over the listing below to see what parts make up what section of the visual.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">

    <title>My page title</title>
    <link href="https://fonts.googleapis.com/css?family=Open+Sans+Condens
    <link rel="stylesheet" href="style.css">

    <!-- the below three lines are a fix to get HTML5 semantic elements w
    <!--[if lt IE 9]>
      <script src="https://cdnjs.cloudflare.com/ajax/libs/html5shiv/3.7.3
    <![endif]-->
```

```
14    </head>
15
16    <body>
17      <!-- Here is our main header that is used across all the pages of our
18
19      <header>
20        <h1>Header</h1>
21      </header>
22
23      <nav>
24        <ul>
25          <li><a href="#">Home</a></li>
26          <li><a href="#">Our team</a></li>
27          <li><a href="#">Projects</a></li>
28          <li><a href="#">Contact</a></li>
29        </ul>
30
31         <!-- A Search form is another commmon non-linear way to navigate t
32
33         <form>
34           <input type="search" name="q" placeholder="Search query">
35           <input type="submit" value="Go!">
36         </form>
37       </nav>
38
39      <!-- Here is our page's main content -->
40      <main>
41
42        <!-- It contains an article -->
43        <article>
44          <h2>Article heading</h2>
45
46          <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Done
47
48          <h3>Subsection</h3>
49
50          <p>Donec ut librero sed accu vehicula ultricies a non tortor. Lor
51
52          <p>Pelientesque auctor nisi id magna consequat sagittis. Curabitu
53
```

```
        <h3>Another subsection</h3>

        <p>Donec viverra mi quis quam pulvinar at malesuada arcu rhoncus.

        <p>Vivamus fermentum semper porta. Nunc diam velit, adipscing ut
      </article>

      <!-- the aside content can also be nested within the main content -
      <aside>
        <h2>Related</h2>

        <ul>
          <li><a href="#">Oh I do like to be beside the seaside</a></li>
          <li><a href="#">Oh I do like to be beside the sea</a></li>
          <li><a href="#">Although in the North of England</a></li>
          <li><a href="#">It never stops raining</a></li>
          <li><a href="#">Oh well...</a></li>
        </ul>
      </aside>

    </main>

    <!-- And here is our main footer that is used across all the pages of

    <footer>
      <p>©Copyright 2050 by nobody. All rights reversed.</p>
    </footer>

  </body>
</html>
```

Take some time to look over the code and understand it — the comments inside the code should also help you to understand it. We aren't asking you to do much else in this article, because the key to understanding document layout is writing a sound HTML structure, and then laying it out with CSS. We'll wait for this until you start to study CSS layout as part of the CSS topic.

HTML layout elements in more detail

It's good to understand the overall meaning of all the HTML sectioning elements in detail — this is something you'll work on gradually as you start to get more experience with web development. You can find a lot of detail by reading our HTML element reference. For now, these are the main definitions that you should try to understand:

- `<main>` is for content *unique to this page.* Use `<main>` only *once* per page, and put it directly inside `<body>`. Ideally this shouldn't be nested within other elements.
- `<article>` encloses a block of related content that makes sense on its own without the rest of the page (e.g., a single blog post).
- `<section>` is similar to `<article>`, but it is more for grouping together a single part of the page that constitutes one single piece of functionality (e.g., a mini map, or a set of article headlines and summaries). It's considered best practice to begin each section with a heading; also note that you can break `<article>`s up into different `<section>`s, or `<section>`s up into different `<article>`s, depending on the context.
- `<aside>` contains content that is not directly related to the main content but can provide additional information indirectly related to it (glossary entries, author biography, related links, etc.).
- `<header>` represents a group of introductory content. If it is a child of `<body>` it defines the global header of a webpage, but if it's a child of an `<article>` or `<section>` it defines a specific header for that section (try not to confuse this with titles and headings).
- `<nav>` contains the main navigation functionality for the page. Secondary links, etc., would not go in the navigation.
- `<footer>` represents a group of end content for a page.

Non-semantic wrappers

Sometimes you'll come across a situation where you can't find an ideal semantic element to group some items together or wrap some content. Sometimes you might want to just group a set of elements together to affect them all as a single entity with some CSS or JavaScript. For cases like these, HTML provides the `<div>` and `<span>` elements. You should use these preferably with a suitable `class` attribute, to provide some kind of label for them so they can be easily targeted.

`<span>` is an inline non-semantic element, which you should only use if you can't think of a better semantic text element to wrap your content, or don't want to add any specific meaning. For example:

```
<p>The King walked drunkenly back to his room at 01:00, the beer doing nc
him as he staggered through the door <span class="editor-note">[Editor's
play, the lights should be down low]</span>.</p>
```

In this case, the editor's note is supposed to merely provide extra direction for the director of the play; it is not supposed to have extra semantic meaning. For sighted users, CSS would perhaps be used to distance the note slightly from the main text.

`<div>` is a block level non-semantic element, which you should only use if you can't think of a better semantic block element to use, or don't want to add any specific meaning. For example, imagine a shopping cart widget that you could choose to pull up at any point during your time on an e-commerce site:

```
<div class="shopping-cart">
  <h2>Shopping cart</h2>
  <ul>
    <li>
      <p><a href=""><strong>Silver earrings</strong></a>: $99.95.</p>
      <img src="../products/3333-0985/thumb.png" alt="Silver earrings">
    </li>
    <li>
      ...
    </li>
  </ul>
  <p>Total cost: $237.89</p>
</div>
```

This isn't really an `<aside>`, as it doesn't necessarily relate to the main content of the page (you want it viewable from anywhere). It doesn't even particularly warrant using a `<section>`, as it isn't part of the main content of the page. So a `<div>` is fine in this case. We've included a heading as a signpost to aid screenreader users in finding it.

> **Warning**: Divs are so convenient to use that it's easy to use them too much. As they carry no semantic value, they just clutter your HTML code. Take care to use them only when there is no better semantic solution and try to reduce their usage to the minimum otherwise you'll have a hard time updating and maintaining your documents.

Line breaks and horizontal rules

Two elements that you'll use occasionally and will want to know about are `<br>` and `<hr>`:

`<br>` creates a line break in a paragraph; it is the only way to force a rigid structure in a situation where you want a series of fixed short lines, such as in a postal address or a poem. For example:

```
<p>There once was a man named O'Dell<br>
Who loved to write HTML<br>
But his structure was bad, his semantics were sad<br>
and his markup didn't read very well.</p>
```

Without the `<br>` elements, the paragraph would just be rendered in one long line (as we said earlier in the course, HTML ignores most whitespace); with `<br>` elements in the code, the markup renders like this:

There once was a man named O'Dell
Who loved to write HTML
But his structure was bad, his semantics were sad
and his markup didn't read very well.

`<hr>` elements create a horizontal rule in the document that denotes a thematic change in the text (such as a change in topic or scene). Visually it just looks like a horizontal line. As an example:

```
<p>Ron was backed into a corner by the marauding netherbeasts. Scared, bu
<hr>
<p>Meanwhile, Harry was sitting at home, staring at his royalty statement
```

Would render like this:

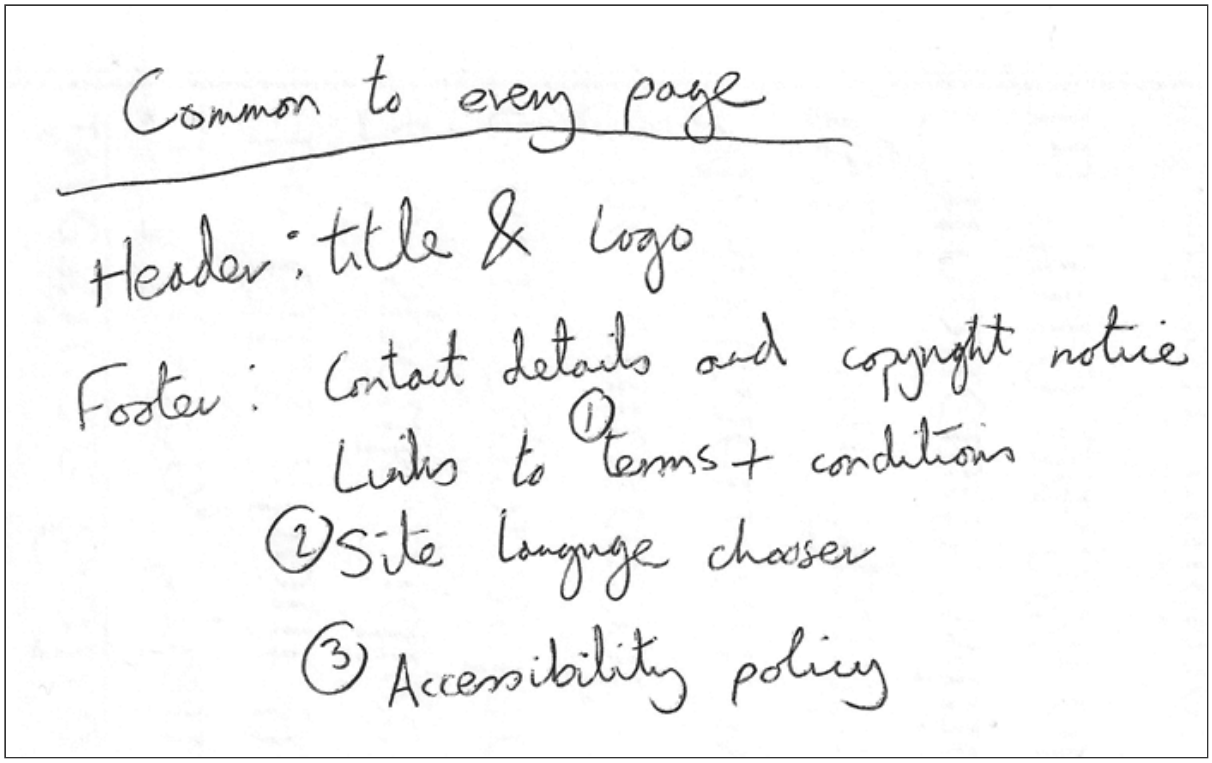
Ron was backed into a corner by the marauding netherbeasts. Scared, but determined to protect his friends, he raised his wand and prepared to do battle, hoping that his distress call had made it through.

Meanwhile, Harry was sitting at home, staring at his royalty statement and pondering when the next spin off series would come out, when an enchanted distress letter flew through his window and landed in his lap. He read it hazily and sighed; "better get back to work then", he mused.

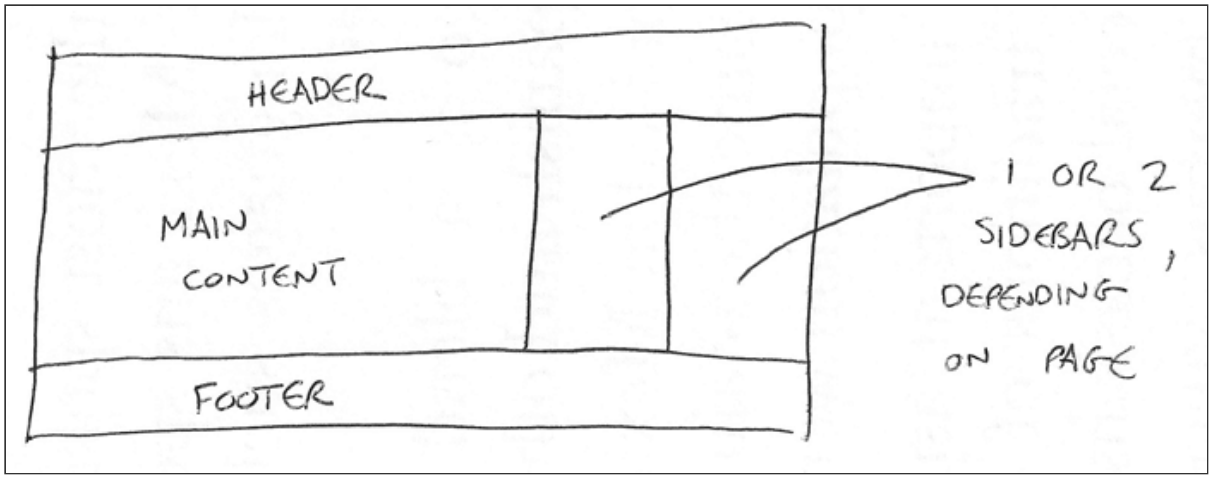
Planning a simple website

Once you've planned out the structure of a simple webpage, the next logical step is to try to work out what content you want to put on a whole website, what pages you need, and how they should be arranged and link to one another for the best possible user experience. This is called Information architecture. In a large, complex website, a lot of planning can go into this process, but for a simple website of a few pages, this can be fairly simple, and fun!

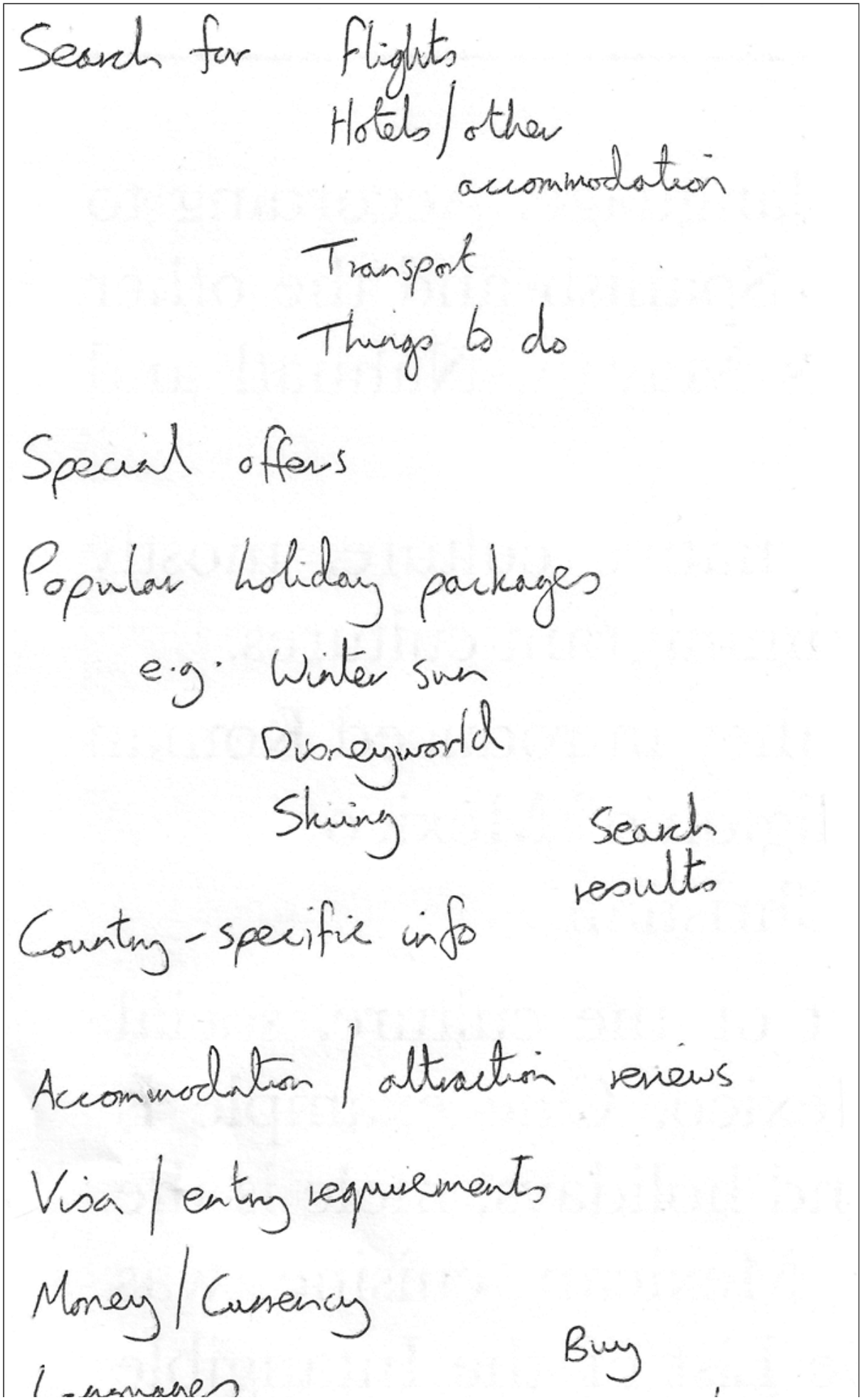
1. Bear in mind that you'll have a few elements common to most (if not all) pages — such as the navigation menu, and the footer content. If your site is for a business, for example, it's a good idea to have your contact information available in the footer on each page. Note down what you want to have common to every page.

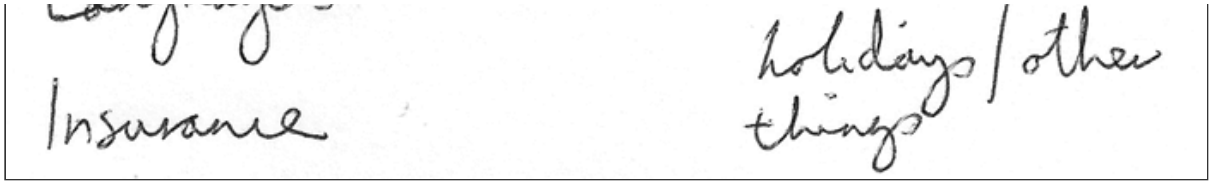


2. Next, draw a rough sketch of what you might want the structure of each page to look like (it might look like our simple website above). Note what each block is going to be.



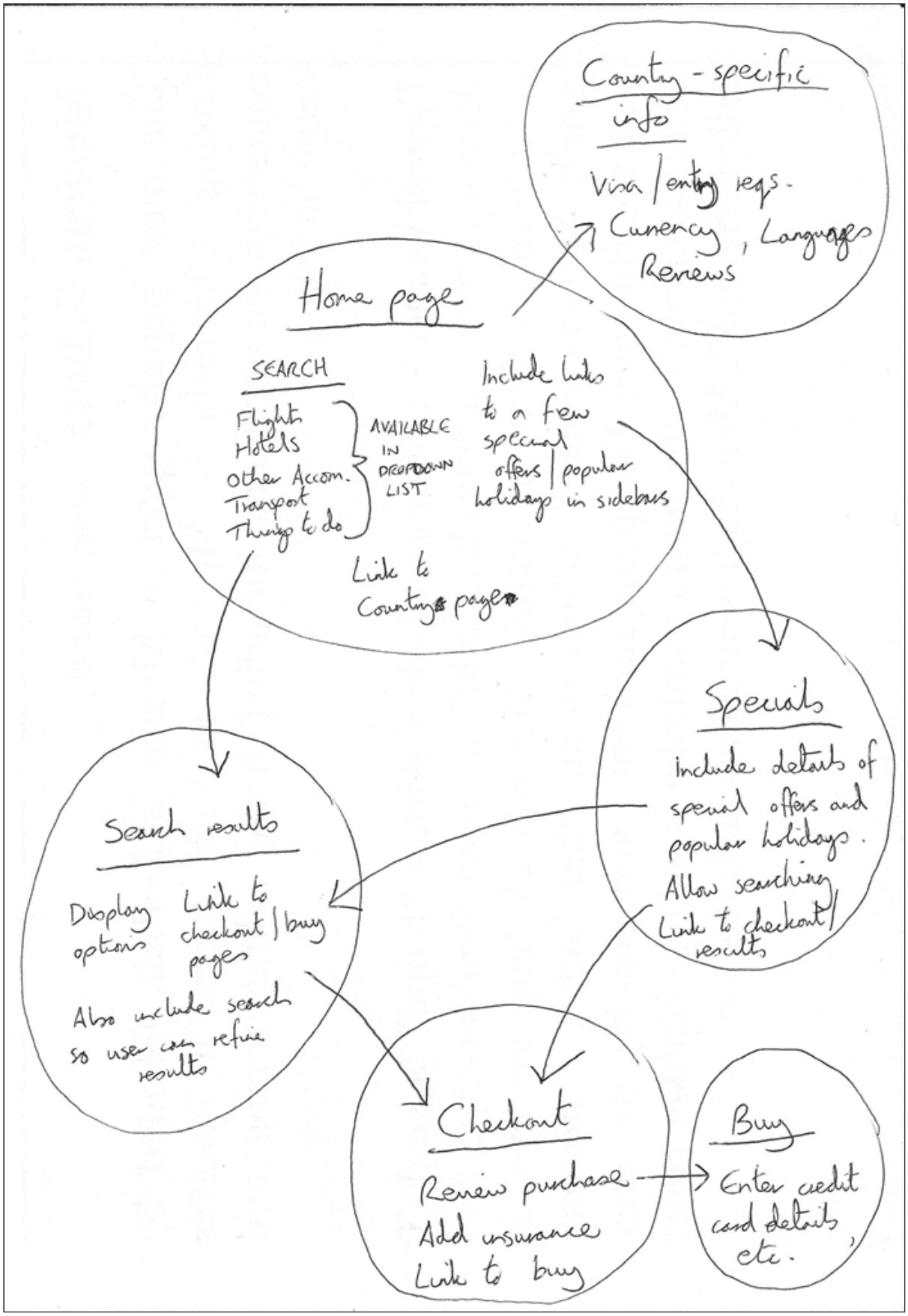
3. Now, brainstorm all the other (not common to every page) content you want to have on your website — write a big list down.


Search for
Flights
Hotels / other accommodation
Transport
Things to do
Special offers
Popular holiday packages
e.g. Winter sun
Disneyworld
Skiing
Search results
Country-specific info
Accommodation / attraction reviews
Visa / entry requirements
Money / Currency
Buy



4. Next, try to sort all these content items into groups, to give you an idea of what parts might live together on different pages. This is very similar to a technique called Card sorting.

5. Now try to sketch a rough sitemap — have a bubble for each page on your site, and draw lines to show the typical workflow between pages. The homepage will probably be in the center, and link to most if not all of the others; most of the pages in a small site should be available from the main navigation, although there are exceptions. You might also want to include notes about how things might be presented.



Active learning: create your own sitemap

Try carrying out the above exercise for a website of your own creation. What would you like to make a site about?

> **Note**: Save your work somewhere; you might need it later on.

Test your skills!

You've reached the end of this article, but can you remember the most important information? You can find a detailed assessment that tests these skills at the end of the module; see Structuring a page of content. We'd advise going through the next article in the series first and not just skipping to it though!

Summary

At this point you should have a better idea about how to structure a web page/site. In the last article of this module, we'll study how to debug HTML.

See also

- Using HTML sections and outlines: Advanced guide to HTML5 semantic elements and the HTML5 outline algorithm.

Previous Overview: Introduction to HTML Next

Debugging HTML

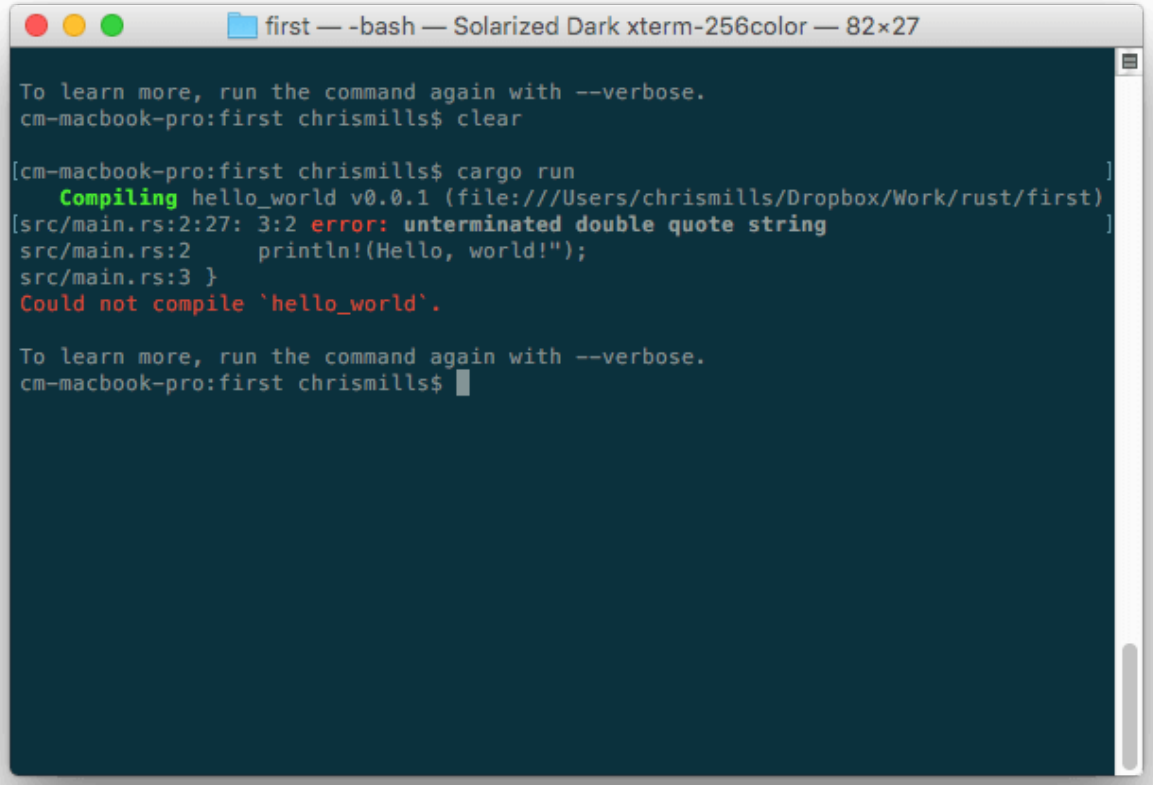
Previous Overview: Introduction to HTML Next

Writing HTML is fine, but what if something goes wrong, and you can't work out where the error in the code is? This article will introduce you to some tools that can help you find and fix errors in HTML.

Prerequisites:	HTML familiarity, as covered in for example Getting started with HTML, HTML text fundamentals, and Creating hyperlinks.
Objective:	Learn the basics of using debugging tools to find problems in HTML.

Debugging isn't scary

When writing code of some kind, everything is usually fine, until that dreaded moment when an error occurs — you've done something wrong, so your code doesn't work — either not at all, or not quite how you wanted it to. For example, the following shows an error reported when trying to compile a simple program written in the Rust language.



Here, the error message is relatively easy to understand — "unterminated double quote string". If you look at the listing, you can probably see how `println!(Hello, world!");` might logically be missing a double quote. However, error messages can quickly get more complicated and less easy to interpret as programs get bigger, and even simple cases can look a little intimidating to someone who doesn't know anything about Rust.

Debugging doesn't have to be scary though — the key to being comfortable with writing and debugging any programming language or code is familiarity with both the language and the tools.

HTML and debugging

HTML is not as complicated to understand as Rust. HTML is not compiled into a different form before the browser parses it and shows the result (it is *interpreted*, not *compiled*). And HTML's element syntax is arguably a lot easier to understand than a "real programming language" like

Rust, JavaScript, or Python. The way that browsers parse HTML is a lot more **permissive** than how programming languages are run, which is both a good and a bad thing.

Permissive code

So what do we mean by permissive? Well, generally when you do something wrong in code, there are two main types of error that you'll come across:

- **Syntax errors**: These are spelling errors in your code that actually cause the program not to run, like the Rust error shown above. These are usually easy to fix as long as you are familiar with the language's syntax and know what the error messages mean.
- **Logic errors**: These are errors where the syntax is actually correct, but the code is not what you intended it to be, meaning that program runs incorrectly. These are often harder to fix than syntax errors, as there isn't an error message to direct you to the source of the error.

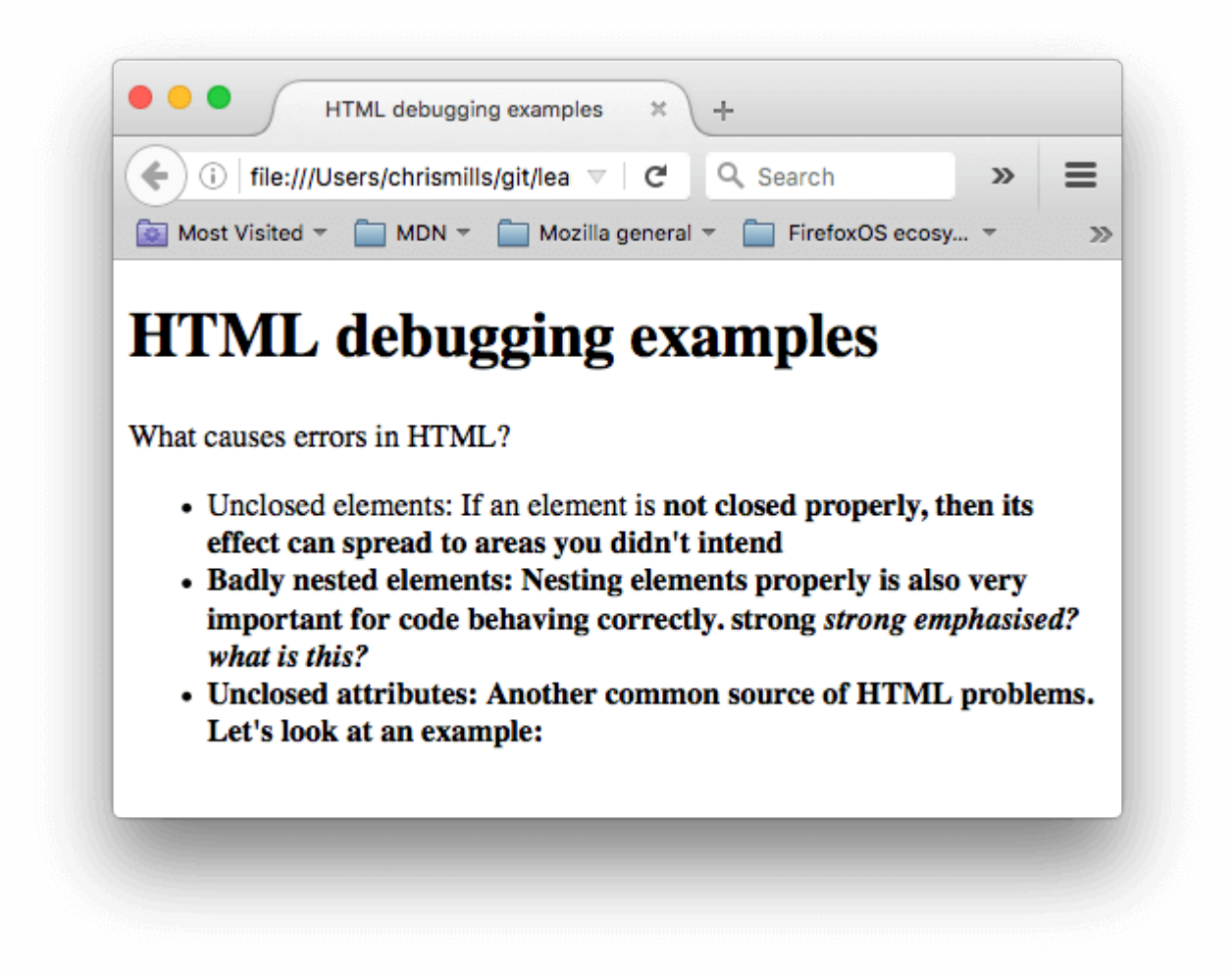
HTML itself doesn't suffer from syntax errors because browsers parse it permissively, meaning that the page still displays even if there are syntax errors. Browsers have built-in rules to state how to interpret incorrectly written markup, so you'll get something running, even if it is not what you expected. This, of course, can still be a problem!

> **Note**: HTML is parsed permissively because when the web was first created, it was decided that allowing people to get their content published was more important than making sure the syntax was absolutely correct. The web would probably not be as popular as it is today, if it had been more strict from the very beginning.

Active learning: Studying permissive code

It's time to study the permissive nature of HTML code.

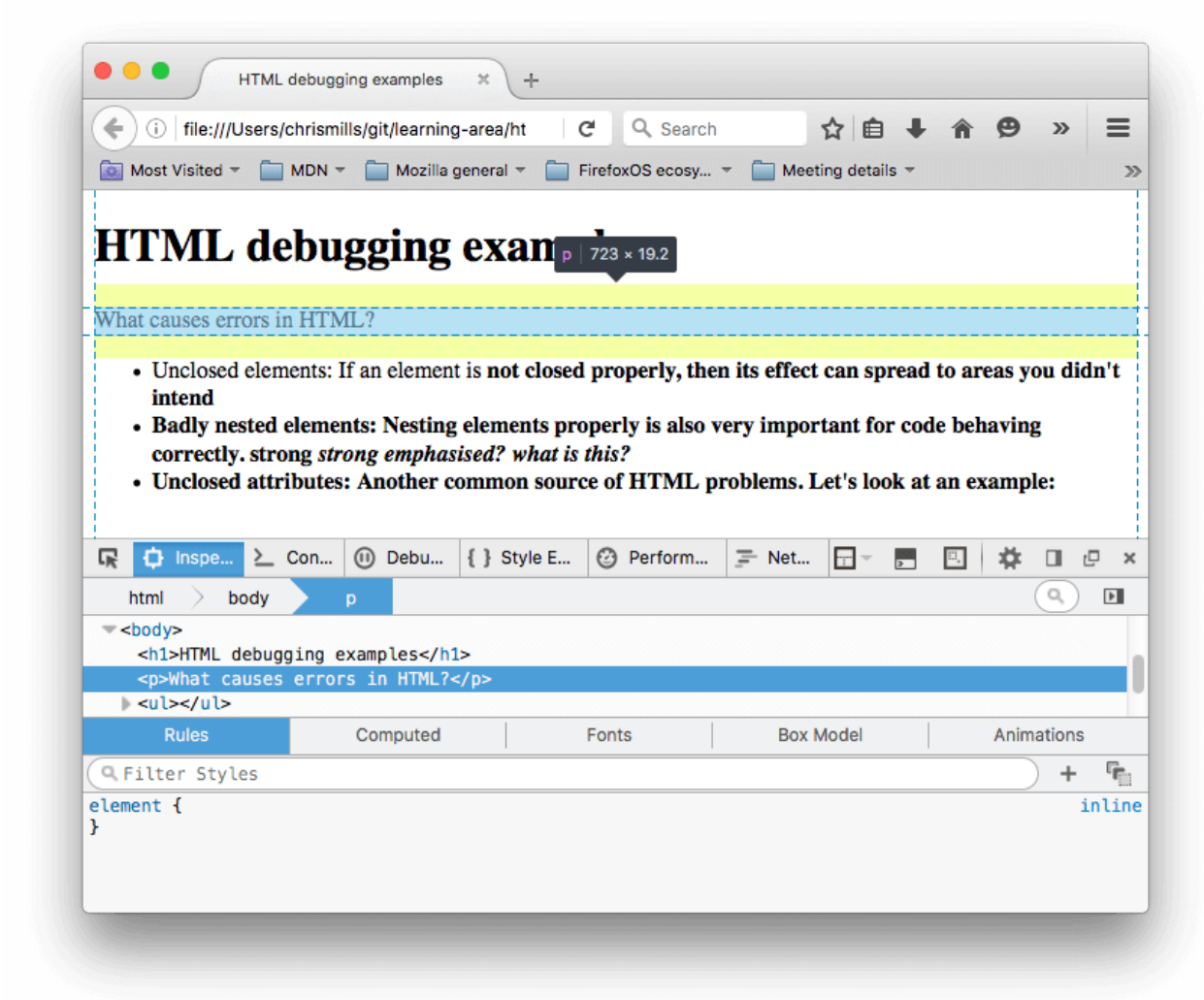
1. First, download our debug-example demo and save it locally. This demo is deliberately written to have some errors in it for us to explore (the HTML markup is said to be **badly-formed**, as opposed to **well-formed**).
2. Next, open it in a browser. You will see something like this:



3. This immediately doesn't look great; let's look at the source code to see if we can work out why (only the body contents are shown):

```
<h1>HTML debugging examples</h1>

<p>What causes errors in HTML?

<ul>
  <li>Unclosed elements: If an element is <strong>not closed prop
      then its effect can spread to areas you didn't intend

  <li>Badly nested elements: Nesting elements properly is also ve
      for code behaving correctly. <strong>strong <em>strong emph
      what is this?</em>

  <li>Unclosed attributes: Another common source of HTML problems
      look at an example: <a href="https://www.mozilla.org/>link
      homepage</a>
</ul>
```

4. Let's review the problems:
    - The paragraph and list item elements have no closing tags. Looking at the image above, this doesn't seem to have affected the markup rendering too badly, as it is easy to infer where one element should end and another should begin.
    - The first `<strong>` element has no closing tag. This is a bit more problematic, as it isn't easy to tell where the element is supposed to end. In fact, the whole of the rest of the text has been strongly emphasised.
    - This section is badly nested: `<strong>strong <em>strong emphasised? </strong> what is this?</em>`. It is not easy to tell how this has been interpreted because of the previous problem.
    - The `href` attribute value has a missing closing double quote. This seems to have caused the biggest problem — the link has not rendered at all.
5. Now let's look at the markup the browser has rendered, as opposed to the markup in the source code. To do this, we can use the browser developer tools. If you are not familiar with how to use your browser's developer tools, take a few minutes to review Discover browser developer tools.
6. In the DOM inspector, you can see what the rendered markup looks like:



7. Using the DOM inspector, let's explore our code in detail to see how the browser has tried to fix our HTML errors (we did the review in Firefox; other modern browsers *should* give the same result):
    - The paragraphs and list items have been given closing tags.
    - It isn't clear where the first `<strong>` element should be closed, so the browser has wrapped each separate block of text with its own strong tag, right down to the bottom of the document!
    - The incorrect nesting has been fixed by the browser like this:

      ```
      <strong>strong
        <em>strong emphasised?</em>
      </strong>
      <em> what is this?</em>
      ```

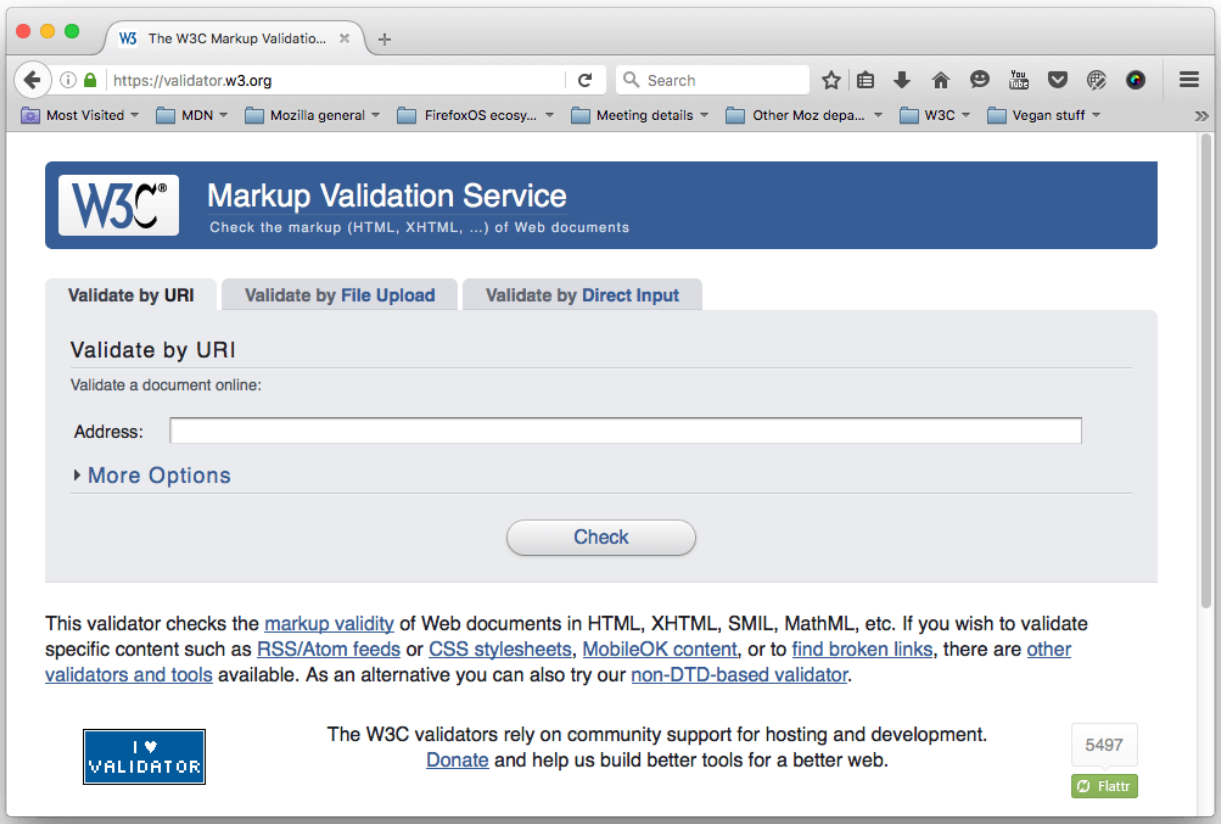
    - The link with the missing double quote has been deleted altogether. The last list item looks like this:

```
<li>
  <strong>Unclosed attributes: Another common source of HTML
  Let's look at an example: </strong>
</li>
```

HTML validation

So you can see from the above example that you really want to make sure your HTML is well-formed! But how? In a small example like the one seen above, it is easy to search through the lines and find the errors, but what about a huge, complex HTML document?

The best strategy is to start by running your HTML page through the Markup Validation Service — created and maintained by the W3C, the organization that looks after the specifications that define HTML, CSS, and other web technologies. This webpage takes an HTML document as an input, goes through it, and gives you a report to tell you what is wrong with your HTML.



To specify the HTML to validate, you can give it a web address, upload an HTML file, or directly input some HTML code.

Active learning: Validating an HTML document

Let's try this with our sample document.

1. First, load up the Markup Validation Service in one browser tab, if it isn't already.
2. Switch to the Validate by Direct Input tab.
3. Copy all the sample document's code (not just the body) and paste it into the large text area shown in the Markup Validation Service.
4. Press the *Check* button.

This should give you a list of errors and other information.

Interpreting the error messages

The error messages are usually helpful, but sometimes they are not so helpful; with a bit of practice you can work out how to interpret these to fix your code. Let's go through the error

messages and what they mean. You'll see that each message comes with a line and column number to help you to locate the error easily.

- "End tag `li` implied, but there were open elements" (2 instances): These messages indicate that an element is open that should be closed. The ending tag is implied, but not actually there. The line/column information points to the first line after the line where the closing tag should really be, but this is a good enough clue to see what is wrong.
- "Unclosed element `strong`": This is really easy to understand — a `<strong>` element is unclosed, and the line/column information points right to where it is.
- "End tag `strong` violates nesting rules": This points out the incorrectly nested elements, and the line/column information points out where it is.
- "End of file reached when inside an attribute value. Ignoring tag": This one is rather cryptic; it refers to the fact that there is an attribute value not properly formed somewhere, possibly near the end of the file because the end of the file appears inside the attribute value. The fact that the browser doesn't render the link should give us a good clue as to what element is at fault.
- "End of file seen and there were open elements": This is a bit ambiguous, but basically refers to the fact there are open elements that need to be properly closed. The lines numbers point to the last few lines of the file, and this error message comes with a line of code that points out an example of an open element:

  ```
  1 example: <a href="https://www.mozilla.org/>link to Mozilla homepa
  ```

  > **Note**: An attribute missing a closing quote can result in an open element because the rest of the document is interpreted as the attribute's content.

- "Unclosed element `ul`": This is not very helpful, as the `<ul>` element *is* closed correctly. This error comes up because the `<a>` element is not closed, due to the missing closing quote mark.

If you can't work out what every error message means, don't worry about it — a good idea is to try fixing a few errors at a time. Then try revalidating your HTML to show what errors are left. Sometimes fixing an earlier error will also get rid of other error messages — several errors can often be caused by a single problem, in a domino effect.

You will know when all your errors are fixed when you see the following banner in your output:

The document validates according to the specified schema(s) and to additional constraints checked by the validator.

Summary

So there we have it, an introduction to debugging HTML, which should give you some useful skills to count on when you start to debug CSS, JavaScript, and other types of code later on in your career. This also marks the end of the Introduction to HTML module learning articles — now you can go on to testing yourself with our assessments: the first one is linked below.

Previous Overview: Introduction to HTML Next

In this module

Marking up a letter

Previous Overview: Introduction to HTML Next

We all learn to write a letter sooner or later; it is also a useful example to test our text formatting skills. In this assignment, you'll have a letter to mark up as a test for your HTML text formatting skills, as well as hyperlinks and proper use of the HTML `<head>` element.

Prerequisites:	Before attempting this assessment you should have already worked through Getting started with HTML, What's in the head? Metadata in HTML, HTML text fundamentals, Creating hyperlinks, and Advanced text formatting.
Objective:	Test basic and advanced HTML text formatting, use of hyperlinks, and use of HTML <head>.

Starting point

To begin, get the raw text you need to mark up, and the the CSS to style the HTML. Create a new `.html` file using your text editor or use an online tool such as CodePen, jsFiddle, or Glitch to complete the tasks.

> **Note**: If you get stuck, then ask us for help. See the Assessment or further help section at the bottom of this page.

Project brief

For this project, your task is to mark up a letter that needs to be hosted on a university intranet. The letter is a response from a research fellow to a prospective PhD student concerning their application to the university.

Block/structural semantics

- Use appropriate document structure including doctype, and `<html>`, `<head>` and `<body>` elements.
- In general, the letter should be marked up as an organization of headings and paragraphs, with the following exception. There is one top level heading (the "Re:" line) and three second level headings.
- Use an appropriate list type to mark up the semester start dates, study subjects, and exotic dances.
- Put the two addresses inside `<address>` elements. Each line of the address should sit on a new line, but not be in a new paragraph.

Inline semantics

- The names of the sender and receiver (and *Tel* and *Email*) should be marked up with strong importance.
- The four dates in the document should be have appropriate elements containing machine-readable dates.
- The first address and first date in the letter should have a class attribute value of *sender-column*. The CSS you'll add later will cause these to be right aligned, as it should be in the case in a classic letter layout.
- Mark up the five acronyms/abbreviations in the main text of the letter to provide expansions of each acronym/abbreviation.
- The six sub/superscripts should be marked up appropriately — in the chemical formulae, and the numbers 103 and 104 (they should be 10 to the power of 3 and 4, respectively).
- Try to mark up at least two appropriate words in the text with strong importance/emphasis.

- There are two places where the letter should have a hyperlink. Add appropriate links with titles. For the location that the links point to, you may use http://example.com as the URL.
- Mark up the university motto quote and citation with appropriate elements.

The head of the document

- The character set of the document should be set as utf-8 using the appropriate meta tag.
- The author of the letter should be specified in an appropriate meta tag.
- The provided CSS should be included inside an appropriate tag.

Hints and tips

- Use the W3C HTML validator to validate your HTML. Award yourself bonus points if it validates.
- You don't need to know any CSS to do this assignment. You just need to put the provided CSS inside an HTML element.

Example

The following screenshot shows an example of what the letter might look like after being marked up.

Dr. Eleanor Gaye
Awesome Science faculty
University of Awesome
Bobtown, CA 99999,
USA
Tel: 123-456-7890
Email: no_reply@example.com

20 January 2016

Miss Eileen Dover
4321 Cliff Top Edge
Dover, CT9 XXX
UK

Re: Eileen Dover university application

Dear Eileen,

Thank you for your recent application to join us at the University of Awesome's science faculty to study as part of your PhD next year. I will answer your questions one by one, in the following sections.

Starting dates

We are happy to accomodate you starting your study with us at any time, however it would suit us better if you could start at the beginning of a semester; the start dates for each one are as follows:

- First semester: 9 September 2016
- Second semester: 15 January 2017
- Third semester: 2 May 2017

Please let me know if this is ok, and if so which start date you would prefer.

You can find more information about important university dates on our website.

Subjects of study

At the Awesome Science Faculty, we have a pretty open-minded research facility — as long as the subjects fall somewhere in the realm of science and technology. You seem like an intelligent, dedicated researcher, and just the kind of person we'd like to have on our team. Saying that, of the ideas you submitted we were most intrigued by are as follows, in order of priority:

1. Turning H_2O into wine, and the health benefits of Resveratrol ($C_{14}H_{12}O_3$.)
2. Measuring the effect on performance of funk bassplayers at temperatures exceeding 30°C (86°F), when the audience size exponentially increases (effect of $3 \times 10^3 > 3 \times 10^4$.)
3. HTML and CSS constructs for representing musical scores.

So please can you provide more information on each of these subjects, including how long you'd expect the research to take, required staff and other resources, and anything else you think we'd need to know? Thanks.

Exotic dance moves

Yes, you are right! As part of my post-doctorate work, I *did* study exotic tribal dances. To answer your question, my favourite dances are as follows, with definitions:

Polynesian chicken dance

A little known but *very* influential dance dating back as far as 300BC, a whole village would dance around in a circle like chickens, to encourage their livestock or be "fruitful".

Icelandic brownian shuffle

Before the Icelanders developed fire as a means of getting warm, they used to practice this dance, which involved huddling close together in a circle on the floor, and shuffling their bodies around in imperceptably tiny, very rapid movements. One of my fellow students used to say that he thought this dance inspired modern styles such as Twerking.

Arctic robot dance

An interesting example of historic misinformation, English explorers in the 1960s believed to have discovered a new dance style characterised by "robotic", stilted movements, being practiced by inhabitants of Northern Alaska and Canada. Later on however it was discovered that they were just moving like this because they were really cold.

For more of my research, see my exotic dance research page.

Yours sincerely,

Dr Eleanor Gaye

University of Awesome motto: “Be excellent to each other.” -- *Bill S Preston, Esq*

Assessment or further help

If you would like your work assessed, or if you get stuck and want to ask for help:

1. Put your work in an online shareable editor such as CodePen, jsFiddle, or Glitch.
2. Write a post asking for assessment and/or help at the MDN Discourse forum Learning category. Your post should include:
    - A descriptive title such as "Assessment wanted for Marking up a letter".
    - Details of what you have already tried, and what you would like us to do (if you are stuck and need help, or if you want an assessment).
    - A link to the example you want evaluated or need help with, in an online shareable editor (as mentioned in step 1 above). This is a good habit to develo. It's very hard to help someone with a coding problem without seeing their code.
    - A link to the actual task or assessment page, so we can read the exact wording of the relevant question(s).

Previous Overview: Introduction to HTML Next

Structuring a page of content

Previous Overview: Introduction to HTML

Structuring a page of content ready for laying it out using CSS is a very important skill to master, so in this assessment you'll be tested on your ability to think about how a page might end up looking, and choose appropriate structural semantics to build a layout on top of.

Prerequisites:	Before attempting this assessment you should have already worked through the rest of the course, with a particular emphasis on Document and website structure.
Objective:	To test knowledge of web page structures, and how to represent a prospective layout design in markup.

Starting point

To get this assessment started, you should go and grab the zip file containing all the starting assets.

The zip file contains:

- The HTML you need to add structural markup to.
- CSS to style your markup.

- Images that are used on the page.

Create the example on your local computer, or alternatively use an online tool such as CodePen, jsFiddle, or Glitch to work on the tasks.

> **Note**: If you get stuck, then ask us for help — see the Assessment or further help section at the bottom of this page.

Project brief

For this project, your task is to take the content for the homepage of a bird watching website and add structural elements to it so it can have a page layout applied to it. It needs to have:

- A header spanning the full width of the site containing the main title for the page, the site logo, and the navigation menu. The title and logo appear side by side once styling is applied, and the navigation appears below those two items.
- A main content area containing two columns — a main block to contain the welcome text, and a sidebar to contain image thumbnails.
- A footer containing copyright information and credits.

You need to add a suitable wrapper for:

- The header
- The navigation menu
- The main content
- The welcome text
- The image sidebar
- The footer

You should also:

- Apply the provided CSS to the page by adding another `<link>` element just below the existing one provided at the start.

Hints and tips

- Use the W3C Nu HTML Checker to catch unintended mistakes in your HTML, CSS, and SVG — mistakes you might have otherwise missed — so that you can fix them.
- You don't need to know any CSS to do this assessment; you just need to put the provided CSS inside an HTML element.
- The provided CSS is designed so that when the correct structural elements are added to the markup, they will appear green in the rendered page.
- If you are getting stuck and can't envisage what elements to put where, draw out a simple block diagram of the page layout, and write on the elements you think should wrap each block. This is extremely helpful.

Example

The following screenshot shows an example of what the homepage might look like after being marked up.

Assessment or further help

If you would like your work assessed, or are stuck and want to ask for help:

1. Put your work into an online shareable editor such as CodePen, jsFiddle, or Glitch.
2. Write a post asking for assessment and/or help at the MDN Discourse forum Learning category. Your post should include:
    - A descriptive title such as "Assessment wanted for Structuring a page of content".

- Details of what you have already tried, and what you would like us to do, e.g. if you are stuck and need help, or want an assessment.
- A link to the example you want assessed or need help with, in an online shareable editor (as mentioned in step 1 above). This is a good practice to get into — it's very hard to help someone with a coding problem if you can't see their code.
- A link to the actual task or assessment page, so we can find the question you want help with.

Previous Overview: Introduction to HTML

In this module

Last modified: May 10, 2020, by MDN contributors

Related Topics

Complete beginners start here!

- Getting started with the Web