Mozilla Developer Network Notes

Date: 28th March 2020

Abstract: The following document has been provided as additional reading to complement the Colt Steele’s “Web Development Bootcamp” on Udemy. The document aims to cover some of the key topics covered on the Mozilla Developer Network to try and expedite some of the earlier modules in the course. All content can be found on the Mozilla Developer Network which can be found at <https://developer.mozilla.org/>.

Contents

[1. Getting Started With HTML 3](#_Toc37085038)

[1.1. What Is HTML? 3](#_Toc37085039)

[1.2. Anatomy of a HTML Element. 4](#_Toc37085040)

[1.2.1. Nesting Elements 4](#_Toc37085041)

[1.2.2. Block Versus Inline Elements 4](#_Toc37085042)

[1.2.3. Empty Elements 6](#_Toc37085043)

[1.3. Attributes 7](#_Toc37085044)

[1.3.1. Adding Attributes to An Element 8](#_Toc37085045)

[1.3.2. Boolean Attributes 8](#_Toc37085046)

[1.3.3. Omitting Quotes Around Attribute Values 9](#_Toc37085047)

[1.3.4. Single or Double Quotes? 9](#_Toc37085048)

[1.4. Anatomy of an HTML Document 11](#_Toc37085049)

[1.4.1. Adding Some Features to an HTML Document 12](#_Toc37085050)

[1.4.2. Whitespace in HTML 14](#_Toc37085051)

[1.5. Entity Reference: Including Special Characters in HTML 15](#_Toc37085052)

[1.6. HTML Comments 16](#_Toc37085053)

[2. What’s in The Head? Metadata in HTML 17](#_Toc37085054)

[2.1. What is the HTML Head 17](#_Toc37085055)

[2.2. Adding a Title 19](#_Toc37085056)

[2.3. Metadata: The <meta> Element 20](#_Toc37085057)

[2.3.1. Specifying the Document’s Character Encoding 20](#_Toc37085058)

[2.3.2. Adding an Author and Description 20](#_Toc37085059)

[2.3.3. Other Types of Metadata 20](#_Toc37085060)

[2.4. Adding Custom Icons to Your Site 22](#_Toc37085061)

[2.5. Applying CSS and JavaScript to HTML 23](#_Toc37085062)

[2.6. Setting the Primary Language of the Document 25](#_Toc37085063)

[3. HTML Text Fundamentals 26](#_Toc37085064)

[3.1. The Basics: Headings and Paragraphs 26](#_Toc37085065)

[3.1.1. Implementing Structural Hierarchy 26](#_Toc37085066)

[3.1.2. Why Do We Need Structure? 28](#_Toc37085067)

[3.1.3. Why Do We Need Semantics? 29](#_Toc37085068)

[3.2. Lists 31](#_Toc37085069)

[3.2.1. Unordered Lists 31](#_Toc37085070)

[3.2.2. Ordered Lists 32](#_Toc37085071)

[3.2.3. Nesting Lists 33](#_Toc37085072)

[3.3. Emphasis and Importance 34](#_Toc37085073)

[3.3.1. Emphasis 34](#_Toc37085074)

[3.3.2. Strong Importance 34](#_Toc37085075)

[3.3.3. Italic, Bold and Underline 35](#_Toc37085076)

[4. Creating Hyperlinks 37](#_Toc37085077)

[4.1. What Is A Hyperlink 37](#_Toc37085078)

[4.2. Anatomy of A Link 38](#_Toc37085079)

[4.3. Adding Supporting Information with the Title Attribute 39](#_Toc37085080)

[4.4. A Quick Primer On URL’s And Paths 40](#_Toc37085081)

[4.4.1. Document Fragments 41](#_Toc37085082)

[4.4.2. Absolute Versus Relative URL’s 42](#_Toc37085083)

[4.5. Link Best Practices 43](#_Toc37085084)

[4.5.1. Use Clear Link Wording 43](#_Toc37085085)

[4.5.2. Use Relative Links Wherever Possible 44](#_Toc37085086)

[4.5.3. Linking to Non-HTML Resources – Leave Clear Signposts 44](#_Toc37085087)

[4.5.4. Use the Download Attribute When Linking to a Download 45](#_Toc37085088)

[4.6. Active Learning: Creating a Navigation Menu 46](#_Toc37085089)

[4.7. E-Mail Links 46](#_Toc37085090)

[4.7.1. Specifying Details 47](#_Toc37085091)

[5. Advanced Text Formatting 48](#_Toc37085092)

[5.1. Description Lists 48](#_Toc37085093)

[5.2. Quotations 50](#_Toc37085094)

[5.2.1. Blockquotes 50](#_Toc37085095)

[5.2.2. Inline Quotations 50](#_Toc37085096)

[5.2.3. Citations 51](#_Toc37085097)

[5.3. Abbreviations 52](#_Toc37085098)

[5.4. Marking Up Contact Details 52](#_Toc37085099)

[5.5. Superscript and Subscript 53](#_Toc37085100)

[5.6. Representing Computer Code 54](#_Toc37085101)

[5.7. Marking up Times and Dates 55](#_Toc37085102)

[6. Document and Website Structure 56](#_Toc37085103)

[6.1. Basic Sections of a Document 56](#_Toc37085104)

[6.2. HTML For Structuring Content 58](#_Toc37085105)

[6.3. HTML Layout Elements in More Detail 59](#_Toc37085106)

[6.3.1. Non-Semantic Wrappers 59](#_Toc37085107)

[6.3.2. Line Breaks and Horizontal Rules. 61](#_Toc37085108)

[6.4. Planning A Simple Website 62](#_Toc37085109)

[6. 63](#_Toc37085110)

# Getting Started With HTML

## What Is HTML?

HTML is NOT a programming language. It is a markup language that instructs the browser what content to show, and how it is to be structured.

The language itself depends on structuring content using a series of **elements** in order to enclose content and define their structure or function on the page. This is done via tags.

The MDN gives the following example. Take the following content:

*“My Cat Is Very Grumpy”*

In HTML, we can create a paragraph from this by writing the following in HTML:

<p>My Cat Is Very Grumpy</p>

Output:

My Cat Is Very Grumpy

**NOTE: Tags in HTML are case insensitive. For example, the title tag can be written as <title> or <TITLE> or <Title> or <TiTlE> and still operate correctly as HTML. HOWEVER all lower case is considered the standard practice for HTML.**

## Anatomy of a HTML Element.

HTML elements can be broken down into three components:

* **The Opening Tag (<>)**: Which consists of the name of the element opened and closed in angled brackets. This states where the element begins, and what content it’s effect is being applied to.
* **The Closing Tag (</>)**: Operate in a similar way to the opening tag, but includes a forward slash before the element name. This defines where the element ends. Failing to include the closing tag can create formatting errors on other content as it signifies where the effects of an opening tag should cease.
* **The Content**: The content of an element. More often than not, this will be text.
* **The Element:** The combination of all the aforementioned components.

Figure 1- Overview of a HTML Element. (Image Sourced From: https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction\_to\_HTML/Getting\_started)

### Nesting Elements

Elements can be nested within other elements. Using the previous grumpy cat example, we can say that the cat is **very** grumpy by nesting the content with an inner set of tags. For example:

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

Output:

My cat is **very** grumpy.

Elements need to be closed in the order of which they appear. For example, the <strong> tag needs to be closed off before the <p> tag as it is the inner most element. Failing to do so results in the browser attempting to best guess what you were trying to implement, which doesn’t always yield the correct results. The following example demonstrates **INCORRECT** nesting:

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

### Block Versus Inline Elements

Elements in HTML can be separated into two categories. These are:

* **Block Level Elements**: These elements form a visible block on the page. The appear on a new line from the previous content stated in the HTML. Any content that follows on from a block element will also be placed on it’s own new line. These are very often the structural elements that represent things such as paragraphs, lists, navigation menus, footers so on so forth. Block level elements **can be nested inside** **other block level elements** but **can’t be nested inside inline elements!**
* **Inline Elements:** Elements contained within block-level elements which surround only a small portion of content. These will not form entire paragraphs and other groupings of content. Inline elements do not cause content to be placed on a newline, they simply carry on from the place they are positioned in text. Examples include the <a> (hyperlink) element, the emphasis <em> element or the bold <strong> element.

The following example demonstrates both types of element.

Inline Example:

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

Output: firstsecondthird

Block Example:

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

Output:

fourth

fifth

sixth

**NOTE: HTML5 has redefined the element categories. These definitions are more accurate and less ambiguous but slightly more complicated to understand than just “block” and “inline”.**

**NOTE: “block” and “inline” should not be confused with the CSS counterparts. Changing the CSS display type does not change the category of the element and doesn’t affect which elements it can contain or which elements it can be contained in. This was fundamentally one of the reasons why these terms were dropped from HTML 5, as they were often a source of confusion.**

### Empty Elements

Empty elements are elements that are made up of a single tag only. They are not followed by a closing tag. These elements are usually used to insert or embed something into the document at the particular place it is included. A good example is the <img> element, which embeds and image file onto a page at the position it is included in. For example:

<img src =<https://raw.githubusercontent.com/mdn/beginner-html-site/ghpages/images/firefox/icon.png>>

Output:



## Attributes

Attributes can be assigned to elements in order to assign extra information to the element that isn’t to appear in the actual content. For example, using the class attribute allows you to give the element a name which can then be used later to target the element with a particular style in CSS.

An attribute should have the following:

* 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.

The following example from the MDN demonstrates this:



Figure 2- An example of an attribute. (Imaged Sourced From: https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction\_to\_HTML/Getting\_started)

### Adding Attributes to An Element

An example of an element that can support attributes is <a>. This is the “anchor” element and allows any wrapped text to become a hyperlink. This element takes many attributes, however some of the more commonly used are:

* **“href”**: The href attribute specifies the web address that you want the link to point to; or where the browser navigates too upon clicking the element. For example, “href=<https://ww.mozilla.org/>”.
* **“title”**: The title attribute specifies extra information about the link such as what page the link sends the user to. For example, “title=”The Mozilla Homepage”. This will appear as a tooltip when the user hovers over the link on a website.
* **“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. To display the link in the current tab, just omit this attribute.

The example below demonstrates how to use the <a> tag to set up a link that displays in a new browser tab:

<p><a href="https://www.bbc.co.uk" target="\_blank">A link</a> to my favorite website.</p>

### Boolean Attributes

Sometimes attributes don’t have values. This is allowed. These are called Boolean attributes, and they can only have one value, which is generally the same as the attribute name. An example of this is the “disabled” attribute, which can be assigned to form input elements. These can be greyed out (i.e. If you don’t want the user to use them)

For example:

<input type=”text” disabled=”disabled>

Alternatively, the above can be written as shorthand as follows:

<!—Using the disabled attribute prevents the end user from entering text into the input box -->

<input type=”text” disabled>

<!—The user can enter text into the following input as it doesn’t contain the disabled attribute -->

<input type=”text”>

### Omitting Quotes Around Attribute Values

Some markup style use attribute values without quotes. In simple elements with single attributes, this works fine, however it creates all sorts of issues when used in multi-attribute elements. **IT IS BEST TO AVOID THIS PRACTICE ALTOGETHER.**

An example of this style is as follows:

<a href=https://www.mozilla.org/>favourite website</a>

Using no quotes in this instance works fine, however, when we add a title to the anchor, we start to encounter issues:

<a href=https://www.mozilla.org/ title=The Mozilla homepage>favourite website</a>

In the above example, the browser misinterprets the mark-up. It thinks the title attribute is three attributes. A title attribute with the value “The”, and two Boolean attributes, Mozilla and homepage. HTML often tries to help the user by doing what it thinks the user intended and running markup no matter what, however, more often than not, it often assumes incorrectly and causes issues for the user.

### Single or Double Quotes?

In the above examples all attributes are wrapped in double quotes. Single quotes can also be used to a similar effect. There are no consequences for opting for either style, it all boils down to a matter of personal coding style. Both 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>

Quotes should, however, not be mixed, as this causes errors. The following example yields errors:

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

In HTML, if the user has used one type of quotation mark to enclose an attribute value, the other type can then be used as part of the value itself. For example:

<a href=<http://ww.example.com> title=”Isn’t This Fun?”> A link to my example.</a>

Including a quotation mark inside an attribute where the same type of quotation mark has been used to enclose the attribute value will result in errors as the value is delimited prematurely as the quote inside the value is treated as a delimiter, not as part of the string. For example:

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

Instead, HTML entities (special values used to denote the presence of a special character) must be used instead. For example, a single quote can be displayed as part of the value using the following code:

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

The code used is then recognised and the appropriate character is inserted as text.

## Anatomy of an HTML Document

The previous section covered what individual elements do, and how they are implemented in HTML. The following section focuses on how these elements can be combined together to produce pages. The following is an example of a basic webpage:

<!DOCTYTPE 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>**: This element is known as the Doctype and has very limited use in modern HTML documents, other than so far as to be required when producing them to make sure everything runs properly. The Doctype originated from the early days of HTML programming, where the element would include a link to a set of rules that the HTML page had to follow in order to be considered good HTML. They often provided automatic error checking and used to look something like:

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

However, these days doctypes are merely an artefact from days gone by. The only thing worth noting is that <!DOCTYPE html> is the shortest string of characters that counts as a valid Doctype.

1. **<html></html>**: The <html> element wraps all content of the entire page. It is known as the **root element**.
2. **<head></head>**: The <head> element acts as a container for all the stuff to be included on the webpage that isn’t content to be shown to the user. This includes things such as keywords, a page description that appears in search results, CSS to style content, character set declarations and more.
3. **<meta charset=”utf-8”>**: This element specifies the character set to be used for the document. In the above instance, it’s the “utf-8” character set which includes most of the characters from most human written languages. It can handle any textual content inputted onto a page with this element embedded in the head. As such, there is usually no reason not to set this, and it generally helps avoid problems further down the road.
4. **<title></title>**: The <title> element specifies the title of the page. This is the string that appears in the browser tab the page is loaded in, and is used to describe the page when a user bookmarks or favourites it.
5. **<body></body>**: The body element contains **ALL** the content that is to be shown to a web user accessing the page. This includes text, images, videos, games, audio or whatever else is to be delivered to the end user.

### Adding Some Features to an HTML Document

The following example details the basic layout of a 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>

The above code has been rewritten to produce a webpage with slightly more detailed content:

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>My test page</title>

</head>

<body>

<h1>Jake's Page</h1>

<p><strong>Welcome to Jake's page.</strong> This web page has been produced as part of the <a href="https://developer.mozilla.org/en-US/" title="The Mozilla Developer Network Webpage" target="\_blank">Mozilla Developer Network's</a> tutorial series on HTML.</p>

<p>Here Is A Bob Ross Meme:</p>

<img src ="https://cdn.acidcow.com/pics/20181109/bob\_ross\_07.jpg">

</body>

</html>

### Whitespace in HTML

The HTML parser will reduce all space separating text down to a single space. This means the developer can insert large amounts of space, or space their code however they like, and the HTML file will still be readable. Tabs are nice ways of formatting HTML and ensuring the code is as readable as possible.

For example, the following markup is valid, albeit, badly formatted:

<p>Dogs are silly.</p>

<p>Dogs are

silly.</p>

## Entity Reference: Including Special Characters in HTML

In HTML, the characters <, >, “, ‘ and & are all special characters which are included in the HTML syntax to perform some kind of operation. Trying to include them into string literals as they are normally, results in the characters being implemented as code and not text.

To get around this, HTML supports character references. These are special codes that represent characters and can be represented by and ampersand, followed by the character code, which is then terminated by a semicolon (;).

The following list defines the literal character and their equivalent character reference:

|  |  |
| --- | --- |
| Literal Character | Character Reference Equivalent |
| < | &lt; |
| > | &gt; |
| “ | &quot; |
| ‘ | &apos; |
| & | &amp; |

Character references closely resemble the name of their literal character representation. More examples can be found on the HTML Wiki.

The following examples demonstrates how to use character references. The first example is incorrect and results in a new paragraph element wrapping “element”. The second line is correct and outputs “<p>” in the text:

<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>

## HTML Comments

HTML supports the use of comments. These are notes written by the programmer which allow them to annotate their markup and explain how it functions without being processed by the browser.

Comments can be implemented by using “<!—( comment) -->”.

The following is an example of how to implement comments:

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

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

The first line of code will be read by the browser and output a paragraph. The second line will not be read by the browser as it is a comment. No paragraph will be output.

# 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 the CSS (if the content is styled), links to custom favicons, and other metadata (data about the HTML, such as the author, and important keywords that describe the document.). The following section details what the head is and the various elements used within to define the meta-data needed to support webpages.

## What is the HTML Head

Take the previous example given in the last section:

<!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 when the webpage is loaded in a browser), the head’s content is not displayed on the page. Instead, the head’s job is to contain metadata (data that describes data) about the document. The above example shows a document with a small head:

<head>

<meta charset="utf-8">

<title>My test page</title>

</head>

In larger pages, the head can become quite sizable. The following section, however, does not aim to detail every element that can be used in the head. It is there to explain the fundamentals elements that provide the major details and components most users expect when using a website (such as a title for the webpage).

## Adding a Title

As previously mentioned, the <title> element is used to add a title to the document which is displayed in the browser tab. This is NOT the same as the <h1> element, which is used to add a top-level heading to the body of document. Though the <h1> text is often referred to as a page title, it is NOT THE SAME as the <title> element. The differences can be summarised as follows:

* 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 the page content (e.g. The story title, the news headline, the item you’re selling, etc.).
* The <title> element is metadata that represents the title of the overall HTML document (not the content of the document).

The <title> elements contents are also used when bookmarking pages in the browser. The content enclosed in the <title> tags will be used as the bookmark suggested name. It’s also worth noting the <title> value is also used in search results by search engines.

## Metadata: The <meta> Element

Metadata is the data that describes data. In order to properly add this to a HTML document, the HTML syntax uses the <meta> element to distinguish it from the rest of the content in the <head> section. The following section details a few examples of <meta> elements that are commonly found in the <head> section.

### Specifying the Document’s Character Encoding

In the previous full-page example, the following line was used:

<meta charset="utf-8">

This meta element specifies the character encoding – or the character set the document is permitted to use. In this instance, utf-8 is used. This is a universal character set which includes almost all the characters used in the human language. This means the web page should be able to handle displaying almost any language or characters used, and as such, it’s normally a good idea to set the character set as utf-8 by default on most webpages.

Using an alternative set, such as the ISO-8859-1 character set (the Latin alphabet) with Japanese symbols may result in strange artefacts appearing in the text.

### Adding an Author and Description

Many <meta> elements include name and content attributes these elements perform the following purposes:

* “name” specifies the type of meta element it is; what type of information it contains.
* “content” specifies the actual meta content.

Two important examples of meta content that make use of the name and description attributes are the “author” and “description” meta elements. These allow the page to contain a description of who wrote the content for the page, and a concise description of what the page is about. For example:

<meta name="author" content="Chris Mills">

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

Specifying an author is useful in instances where the page audience may wish to know who wrote the page in order to get in touch with the author regarding questions about the piece or about who they are and what they do. In fact, some Content Management Systems (Such as WordPress) have systems that automatically extract page author information and make it available for such purposes.

Specifying a description that includes keywords associated with the content of the page allow the page to become more relevant in search engine searches and thus improve the potential traffic that heads to the site. This is known as “Search Engine Optimisation”.

### Other Types of Metadata

Some other types of metadata elements are proprietary and have been provided so certain sites (such as Facebook or Twitter) can access certain information from the webpage.

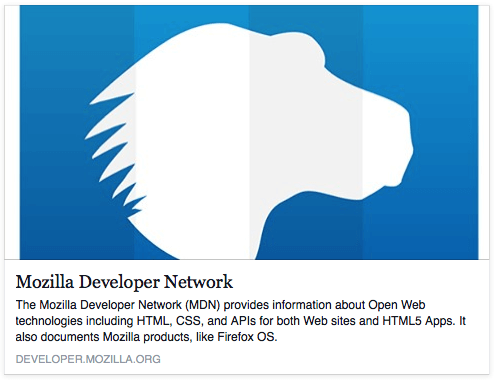
For example, Open Graph Data is a proprietary metadata protocol provided by Facebook that is designed to provide richer metadata for websites that can be used by Facebook when sharing links. For example:

<meta property="og:image" content="https://developer.cdn.mozilla.net/static/img/opengraph-logo.dc4e08e2f6af.png">

<meta property="og:description" content="The Mozilla Developer Network (MDN) provides information about Open Web technologies including HTML, CSS, and APIs for both Web sites and HTML5 Apps. It also documents Mozilla products, like Firefox OS.">

<meta property="og:title" content="Mozilla Developer Network">

When a page that contains the above metadata is shared on Facebook, the link appears along with an image and description as demonstrated below:



Twitter also uses its own proprietary metadata known as “Twitter Cards” which has a similar effect when a URL with Twitter Cards in it’s metadata is shared on Twitter. For example:

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

## Adding Custom Icons to Your Site

Modern browsers support the use of icons which can be displayed in certain contexts. One of the most common uses is the implementation of a favicon (a “favourites icon”), or an icon that is used alongside the text in the “favourites” or “bookmarks” lists in browsers.

The Favicon has been around for some time. It is a 16x16 pixel image used in multiple places across the browser. It can be displayed in the browser tabs containing each open page as well as next to each bookmarked page in the bookmarks panel.

A favicon be added by saving an image as either a .ico file (better for compatibility, will be supported back as far as Internet Explorer 6) or a .gif or .png on the server and using the following code, referencing the images location in the href attribute.

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

Other icon types exist which are supported by different devices such as phones and tablets. The below example demonstrates some of the different formats of icon supported:

<!-- third-generation iPad with high-resolution Retina display: -->

<link rel="apple-touch-icon-precomposed" sizes="144x144" href="https://developer.cdn.mozilla.net/static/img/favicon144.a6e4162070f4.png">

<!-- iPhone with high-resolution Retina display: -->

<link rel="apple-touch-icon-precomposed" sizes="114x114" href="https://developer.cdn.mozilla.net/static/img/favicon114.0e9fabd44f85.png">

<!-- first- and second-generation iPad: -->

<link rel="apple-touch-icon-precomposed" sizes="72x72" href="https://developer.cdn.mozilla.net/static/img/favicon72.8ff9d87c82a0.png">

<!-- non-Retina iPhone, iPod Touch, and Android 2.1+ devices: -->

<link rel="apple-touch-icon-precomposed" href="https://developer.cdn.mozilla.net/static/img/favicon57.a2490b9a2d76.png">

<!-- basic favicon -->

<link rel="shortcut icon" href="https://developer.cdn.mozilla.net/static/img/favicon32.e02854fdcf73.png">

Knowing each of the icon formats isn’t too important at this point. However, knowing they exist is useful and may help clarify others code if they appear there whilst browsing through it.

## Applying CSS and JavaScript to HTML

Most modern websites employ CSS for styling websites and JavaScript for implementing interactive functionality. These are typically included using the <link> and <script> elements respectively.

* The <link> element always goes inside the head of the document. It takes two attributes, rel=”stylesheet”, which indicates the file is the document’s stylesheet, and the href which is the link to the stylesheet file. The example below demonstrates this:

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

* The <script> element does not have to go in the head. Often it is advisable to include any script at the bottom of the documents body (just before the closing </body> tag) in order to ensure all the HTML content has been read by the browser before any JavaScript is applied to it (reading any elements that have not been loaded in via JavaScript results in the browser throwing an error). The following is an example:

<script src=”my-js-file.js”></script>

**NOTE: The <script> element looks like an empty element, but it isn’t. A closing tag must be provided. This is because a developer can choose not to point to an external file via src, but write the script in between the script tags.**

## Setting the Primary Language of the Document

Finally, it’s worth noting that the developer should set the language of the page using the “lang” attribute in the opening tag as shown below:

<html lang=”en-us”>

This is useful for instances where a webpage may need to be localised to specific language. For example, a webpage will be indexed by a search engine far more effectively if the correct language is set (the webpage will appear correctly in language specific results). It’s also useful in terms of accessibility. For example, people with screen readers rely on the language tags to have the correct pronunciation on their device.

Subsections of a webpage can also be set to be recognised in different languages by setting the “lang” attribute in a tag such as a <span> or a <p>. For example:

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

These codes are defined by the ISO 639-1 standard.

# HTML Text Fundamentals

The main purpose of HTML is to give structure and meaning (also known as semantics) to text so it can be displayed properly by the browser. The following section details how HTML is used to structure a page of text by adding headings and paragraphs, emphasising words, creating lists and more.

## The Basics: Headings and Paragraphs

Most of the text sources we read structures texts into headings and paragraphs, regardless of if we read a story, a newspaper, a textbook, a magazine or a report. Structured content makes consuming content a more enjoyable and easier experience.

In HTML, each paragraph is wrapped in a <p> element like so:

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

Each heading must 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 a useful analogy, we will refer to each element as if it were to be used in a story. <h1> would represent the title of the story, <h2> elements would be the title of each chapter, and <h3> elements would represent sub-sections of each chapter, and so on. For example:

<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 on the ...</p>

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

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

<h3>The spectre speaks</h3>

<p>Several more hours had passed, when all of a sudden the spectre sat bolt upright and exclaimed, "Please have mercy on my soul!"</p>

Each element can represent anything provided the hierarchy makes sense. As a general reference, the following are considered the best practice when implementing a hierarchical structure:

* Preferably you should just use a single <h1> per page – This is the top level heading and all others must sit below this in the hierarchy.
* Any further subheadings must follow the order of <h2>, <h3>, <h4> and so on. For example, <h3> should not be used to represent sub-headings, then <h2> be used to represent sub-subheadings. It doesn’t make sense and leads to weird looking pages.
* Of the six headings available, aim to use no more than three per page unless absolutely necessary. Documents with many heading levels are confusing and difficult to navigate. In instances where many headings are required, it may just be best to spread the content over many pages.

### Why Do We Need Structure?

In the context of HTML, structure is needed else the content becomes one homogenous mess of text. This makes pages cluttered, difficult to read and unattractive. The following HTML demonstrates this:

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>Quick hummus recipe</title>

</head>

<body>

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.

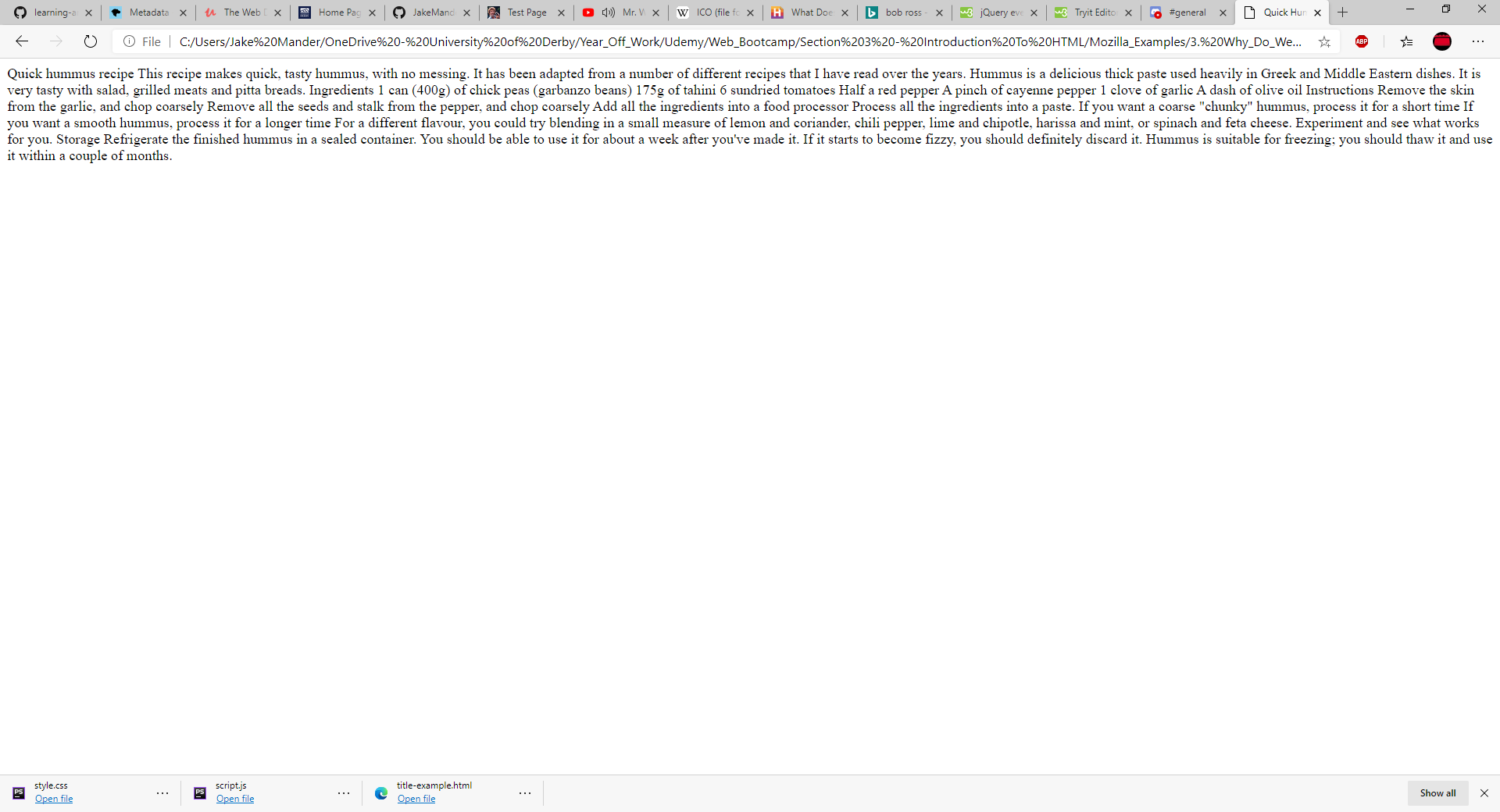
Storage

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.

</body>

</html>

The output of this markup is as follows:

The mess of text that follows is due to the presence of no elements to define any structure to the content. The browser does not know how to structure the text. It does not know what is a heading and what is a paragraph, so it displays everything as a single block. Furthermore:

* Users looking at a web page tend to scan quickly to find relevant content, often just reading the headings to being with. If the user can’t find what they’re looking for within a few seconds, they’ll likely look elsewhere for the content they’re after.
* Search engines indexing your page consider the contents of headings as important. They often contain the keywords that determine the page’s search rankings for a particular search. Without headings, a webpage will perform poorly in terms of SEO (Search Engine Optimisation).
* Severely visually impaired people often don’t read web pages; they listen to them instead with a screen reader. This software provides ways to get fast access to given text content. They also provide outline to documents by reading out the headings and allowing the user to find the content they need quickly. If headings are not available, they will be forced to read the entirety of the document out loud.
* To style content with CSS, or to make it interactive with JavaScript, elements need to wrap the relevant content to provide some form of separation. This allows CSS/JavaScript to more effectively target the content it needs.

Therefore content needs to be structured appropriately.

### Why Do We Need Semantics?

Semantics are relied on in many different areas. 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, however, get very confusing if the correct semantics are not applied (for example, designing a traffic light with the green light meaning “stop”).

Similarly, the same can be said for HTML. The correct elements need to be used in order to give content the correct meaning, function and 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 the page. For example:

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

By default, however, the browser gives it a large font to make it look like a heading (though this can be styled to look like any style the user wants using CSS). More importantly, its semantic value will be used in multiple ways. For example, by the heading will be used by search engines and screen readers as mentioned above.

On the other hand, the developer could make any element look like a top level heading by the following line of code:

<span style="font-size: 32px; margin: 21px 0; display: block;">Is this a top level heading?</span>

This is a <span> element. It has no semantics. It is used to wrap content when a developer wants to apply CSS to a particular inline block of text without giving it any extra meaning. In the example, the span has been made to look like a <h1>, however it will not receive any of the semantic benefits of having been wrapped in a heading tag. This is bad practice. Always try to use the relevant HTML for the right job.

## Lists

Lists are everywhere, and the web is no exception. In HTML there are three main types to learn.

### Unordered Lists

Unordered lists are used to mark up lists of items for which the order of items doesn’t matter. For example, the following shopping list of goods:

milk

eggs

bread

hummus

Every unordered list starts off with a <ul> element – This wraps around all the list items like so:

<ul>

milk

eggs

bread

hummus

</ul>

The last step is to then wrap each item in the list in a <li> (list item) element in order to distinguish each individual item in the list. This can be done like so:

<ul>

<li>milk</li>

<li>eggs</li>

<li>bread</li>

<li>hummus</li>

</ul>

Output:

* milk
* eggs
* bread
* hummus

### Ordered Lists

Ordered lists list should be used when the order of items does matter. For example, a process or a set of directions. For 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 identical to that of unordered lists, except that the list is wrapped inside a pair of <ol> tags, instead of <ul>. For example:

<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>

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

</ol>

### Nesting Lists

HTML allows the developer to nest one list inside another. For example, some cases may require for a list to have sub-bullets sitting below a top level bullet. We can use the hummus example to demonstrate the point:

<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>

<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>

<li>If you want a smooth hummus, process it for a longer time.</li>

</ol>

The last two bullets are closely related to the bullet before them, so it makes sense to nest them inside as sub-bullets in an unordered list as they read like sub-instructions for that specific point. This can be seen in the following 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>

<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 time.</li>

<li>If you want a smooth hummus, process it for a longer time.</li>

</ul>

</li>

</ol>

## Emphasis and Importance

When using the human language, we often emphasise certain words to alter the meaning of a sentence or mark certain words as important or different in context. HTML provides its own set of semantic elements to allow textual context to be marked up with such effects. The following section looks at some of the more common approaches.

### Emphasis

When we want to add emphasis in spoken language, we stress certain words. In doing so, the meaning of what we say is altered. Similarly, in written language, words can be stressed by being written in italics. For example, the following sentences have different meanings.

I am glad you weren't late.

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

The first sentence is carried across as sincere. The meaning is the person is genuinely pleased the individual in question isn’t late. The second sentence however has its meaning changed through the use of emphasis on the words “glad” and “late” via italics. The meaning has become more passive-aggressive and annoyed at someone who is late.

In HTML the <em> (emphasis) element is used to mark up such instances. This can be used to create more interesting pages and helps screen readers speak the page out in the tone it was meant to be read. Browsers style the <em> tag as italic by default, however, it shouldn’t be used to simply get the italic styling. To do that, a <span> element linked to CSS should be used, or alternatively, the content should be wrapped in the <i> tag. This ensures the semantics of the tag are maintained.

An example of using the emphasis tag is as follows:

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

### Strong Importance

To emphasise important words in spoken language, we stress them. In written language, important words are written in **bold**.

In HTML the <strong> (strong importance) is used to mark up such instances. As well as making the content in the document clearer to the reader, it’s also used by screen readers to adjust the tone of voice to better convey the content to the visually impaired.

Browsers often style this as bold by default, but again, developers shouldn’t use the <strong> element to get the bold style if the content does not match up with the semantics of the tag. To do that, the developer should use a span element and use CSS to style the selection, or alternatively, use the <b> tag.

For example:

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

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

Strong and emphasis tags can also be nested inside one another if needed. For example:

<p>This liquid is <strong>highly toxic</strong> —

if you drink it, <strong>you may <em>die</em></strong>.</p>

### Italic, Bold and Underline

The elements above have clear semantic values. It is clear when they should be used when describing content in a web document.

However, the situation with the Bold (<b>), Italics (<i>) and Underline (<u>) elements are slightly more complex. These elements came about in an era where CSS was still poorly supported, or not at all, and enabled developers to clearly highlight sections of text without having to worry about altering the styling. These elements were concerned primarily focused on styling, not semantics and are known as **presentational elements**. The HTML standard now advises that these elements should no longer be used, because they hinder the accessibility and the Search Engine Optimisation (SEO) of the page.

HTML 5 redefined <b>, <i> and <u> with some new semantic roles.

The best rule of thumb is to use <b>, <i> or <u> to convey a meaning traditionally conveyed with bold, italics or underline, assuming there is no more suitable element to do so. Accessibility is key, however, and should not be sacrificed to style content.

Examples of the suitable use of each element are as follows:

* **<i>**: Is used to convey a meaning traditionally covered by italics. For example: Foreign words, taxonomic designation in biology, technical terms, a thought, etc.
* **<b>**: Is used to convey a meaning traditionally conveyed in bold. For example: Key words, product names, lead sentences, etc.
* **<u>**: Is used to convey a meaning traditionally conveyed in underline. For example: Proper name, misspellings, etc.

**NOTE: It is worth noting that people strongly associate underline with hyperlinks on web pages, and as such, it’s use should be avoided unless when semantically appropriate. If possible, use CSS to alter the default underline to something else in order to make it’s use more appropriate and clearer.**

The following example demonstrates the points covered in the above section:

<!-- 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; text-decoration-style: wavy;">spel</u> better.

</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>

# Creating Hyperlinks

Hyperlinks are what makes the web, a web of content. The following section details what syntax makes a link and explains the best practices used to implement a link.

## What Is A Hyperlink

Hyperlinks have been about since the beginning of the web and are the main component that connects the web together. Documents can be linked to other documents (or resources). Specific parts of documents can be linked to. And more recently, apps can be accessed on a web address rather than having to be installed natively on a device. Just about any web content can be converted to a link, so that when clicked (or activated), the browser can go to another web address (URL).

**Note: A URL can point to HTMNL files, text files, images, text documents, video, audio files and anything else that lives on the web. If the web browser doesn’t know how to display or handle the file, it will ask the user if they would like to open or download the file and handle it from there.**

## Anatomy of A Link

A basic link is created by wrapping the text (or other content) you want to turn into a link inside an <a> element and giving it a href attribute (also known as a Hypertext Reference, or Target). This contains the web address the user wants to link the website to.

The following gives an example of implementing a hyperlink:

<p>I'm creating a link to

<a href="https://www.mozilla.org/en-US/">the Mozilla homepage</a>.

</p>

Result: I'm creating a link to [the Mozilla homepage](https://www.mozilla.org/en-US/)

## Adding Supporting Information with the Title Attribute

Another attribute that is useful to add to links is “title”. This contains supplementary useful information about the embedded link such as what information the linked page contains, or other 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>

The title text only appears when the user hovers over the link. If the title contains important information, it may be prudent to include it as part of the web page’s text, as not all users will be using a mouse and keyboard to view the content.

## A Quick Primer On URL’s And Paths

To fully understand link targets, it’s useful to be familiar with URL’s and file paths. This section gives the information required to achieve this.

A URL (or Uniform Resource Locator) is 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/>.

URL’s use paths to find files. Paths specify where in the filesystem the file being requested is located. Mozilla use the following example:

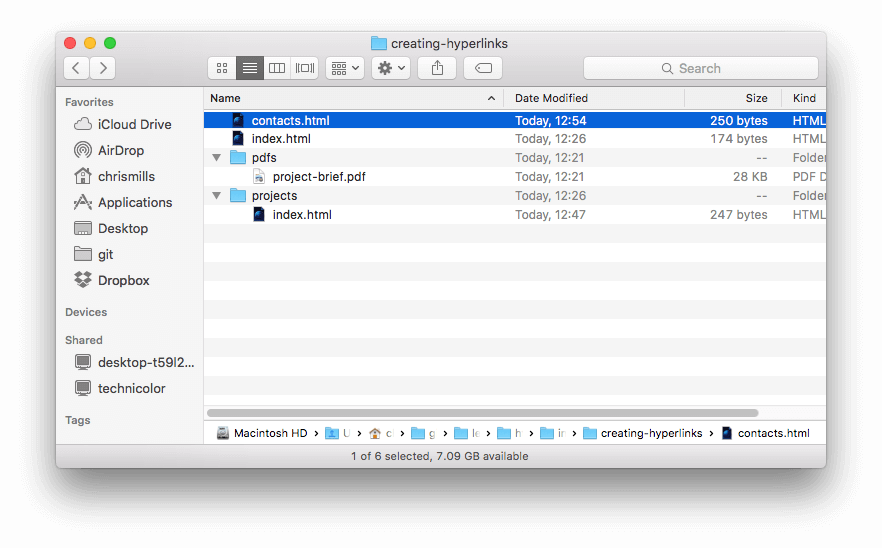


Figure 3- Demonstration of A Website File Structure (Sourced From: https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction\_to\_HTML/Creating\_hyperlinks)

The root of the 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 this directory, there are a couple of html files, index.html and contacts.html. On a real website, index.html Is the home page (or landing page) which serves as the entry point for the website as a whole.

In addition to this, there are two additional directories inside the root, “pdfs” and “projects”. These each have their own single file inside them – a PDF (project-brief.pdf), and an index.html file respectively.

The web directory structure quite happily allows for the website to have more than one index providing they are located in different file directories within the file system. Many websites do this, and in the above example, it may be implemented to be the main landing page for project-related information.

The following examples demonstrate using the file structure of the server and hyperlinks to access documents local to the file on a web server:

* **Same Directory**: Including hyperlinks inside index.html (the top level index.html) pointing to contacts.html, the hyperlink only needs to specify the name of the file to set up the link, this is because, relative to the file the link is being set up in, the destination file is located in the same directory. So the URL required is as follows

<p>Want to contact a specific staff member?

Find details on our <a href="contacts.html">contacts page</a>.</p>

* **Moving Down into Subdirectories**: To link to files located in directories lower than the directory the index.html file is stored, the directory/s must be listed in the path provided to the hyperlink. For example, if we want to access the index.html file in the projects directory from the root index.html file, the following hyperlink must be provided:

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

* **Moving Back Up into Parent Directories:** To provide a hyperlink that targets a resource located in a directory in a directory in a higher-level directory relative to the file holding the link, the link must append two dots (“..”) to represent a higher level. For example, to provide a link to the “pdfs/project-brief.pdf” file from the “index.html” file found in the “projects” directory, the browser needs to move up a directory back into the root, then back down through the “pdfs” directory to find the “project-brief.pdf” file. The following example demonstrates the markup used to implement the link:

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

**NOTE: Multiple instances of these features can be combined to form complex URL’s. For example:**

../../../complex/path/to/my/file.html

### Document Fragments

HTML allows the developer to link to a specific part of a HTML document (known as a document fragment), rather than just to the top of the document. To do this, an id attribute must be assigned to the element the developer wishes to link to. Usually, developers will link to individual headings to access specific sections of content. An example of this could be:

<h2 id="Mailing\_address">Mailing address</h2>

To link to that id, the developer includes the attribute value at the end of the target URL, preceded by a hash/pound symbol. For example:

<p>Want to write us a letter? Use our <a href="contacts.html#Mailing\_address">mailing address</a>.</p>

A document fragment reference can be used on its own to reference a specific part of the same document. This can be done by referencing the id attribute of the element in the href attribute without any other information. For example:

<p>The <a href="#Mailing\_address">company mailing address</a> can be found at the bottom of this page.</p>

### Absolute Versus Relative URL’s

The two key terms to remember regarding the types of URL used are **absolute URL** and **relative URL**. The definitions for both types are as follows:

* **Absolute URL:** Points to a location defined by its absolute location on the web, including the protocol and the domain name. So for example, if an index.html page is uploaded to a directory called “projects” that sits inside the toot 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 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, much in the same manner that was demonstrated 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.exmaple.com/projects/pdfs/project-brief.pdf>).

A relative URL points to different places depending on the actual location of the file you refer from – for example if we moved “index.html” 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”.

In this example, the location of project-brief.pdf file and the pdfs file does not change with the file that uses the relative link. This means the link is now pointing to the wrong place and won’t work when clicked on.

## Link Best Practices

The following section demonstrates good practices to follow when setting up links in HTML.

### Use Clear Link Wording

Links need to be accessible to all readers using a web page, regardless of the current context and the tools used to access the page. 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’s a good idea to include keywords in link text to effectively describe what is being linked to.
* Visual readers skim web pages rather than reading every word. Their eyes are drawn to features that stand out, like links. Descriptive link text is useful in these instances.

The following examples demonstrate clear and unclear links:

**Good Link Text:**

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

Download Firefox

</a></p>

**Bad Link Text:**

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

Click here

</a>

to download Firefox</p>

Other good practices to follow are:

* Don’t repeat the URL as part of the link text – URLs looks ugly and sound even uglier when a screen reader reads them out by letter.
* Don’t say “link” or “links to” in the link text – Doing so creates 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 shouldn’t normally be broken as users are used to the style for representing links).
* Link labels should be as short as possible. Long links annoy screen reader users who have to hear the whole thing read out.
* Minimise 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” and “click here” is confusing.

### Use Relative Links Wherever Possible

It is advisable to avoid absolute links and use relative links whenever possible. Though absolute links rarely break when the host file is moved to an alternative location, relative links provide the following benefits:

* For a start, it is a lot easier to scan HTML markup when relative URLs have been used as they are generally a lot shorter than absolute URLs, which makes reading code easier.
* It is more efficient to use relative URLs wherever possible. When an absolute URL is used, the browser has to look up the real location of the server on the Domain Name System (DNS), then goes to that server to find the file that is being requested. With a relative URL, the browser just looks up the file that is being requested, on the same server. Using absolute URLs when relative URLs suffice creates extra work for the browser, resulting in slower performance.

### 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)m you should add clear wording to reduce any confusion. It can be irritating for users when, for example:

* They are using a low bandwidth connection, click a link, and then a multiple megabyte download starts unexpectedly
* The user hasn’t got Flash Player installed, clicks a link, and then suddenly gets taken to a page that requires Flash.

The following examples demonstrated how text can be used to sign post links properly:

<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 linking to a resource that is to be downloaded rather than opened in the browser, the download attribute can be used to provide a useful, default save filename. The following HTML demonstrates the download attribute used to provide a default filename for the Mozilla Firefox installer:

<a href=<https://download.mozilla.org/?product=firefox-latest-ssl&os=win64&lang=en-US> download="firefox-latest-64bit-installer.exe">

Download Latest Firefox for Windows (64-bit) (English, US)

</a>

## Active Learning: Creating a Navigation Menu

The following markup has been implemented to create a functioning nav menu that uses relative links to function:

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>Projects</title>

</head>

<body>

<ul>

<li>

<a href="index.html" title="A Link Home">Home</a>

</li>

<li>

<a href="pictures.html" title="A Link To The Pictures Page"> Pictures</a>

</li>

<li>

<a href="social.html" title="A Link To How You Can Get In Touch With Me">Social</a>

</li>

</ul>

<h1>My projects</h1>

<p>Welcome to my project page, showing what exciting things I am currently doing.</p>

</body>

</html>

## E-Mail Links

HTML allows for 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>

The above markup results in a link that loads the clients default email client with a new message ready to be written to the desired recipient.

The email address is even optional. If left out (i.e. the href is simply “<mailto:>”), a new outgoing email will be created with the recipient field left blank. This will often be used for “Share” links that users can click on and share the content to an email of their choosing.

### Specifying Details

In addition to the email address, other information can be provided in the href. For example, any standard mail header fields can be added to the “mailto” URL that is provided. The most common of these are the “subject”, “cc” and “body” fields (body is not a true header field, but it allows the user to specify a short content message for the new email to be applied when the email client is loaded). Each filed and its value is specified as a query term.

The following example includes a cc, bcc, subject and body:

<a href="mailto:nowhere@mozilla.org?cc=name2@rapidtables.com&bcc=name3@rapidtables.com&subject=The%20subject%20of%20the%20email&body=The%20body%20of%20the%20email">

Send mail with cc, bcc, subject and body

</a>

**NOTE: The values of each field must be URL-encoded. That is, non-printing characters (i.e. characters such as space, tab, page breaks and character returns) and spaces must be percent-escaped. In addition, the use of the question mark (?) to separate the main URL from the field values, and the ampersand (&) to separate each field should also be noted.**

# Advanced Text Formatting

There are many other elements in HTML for formatting text that were not discussed in the HTML text fundamentals section. The elements in this section are less known, but useful to know about. In this section, quotations, description lists, computer code and other related text, subscript, super script, contact information, and more will be covered.

## Description Lists

In HTML text fundamentals, marking up basic lists (ordered and unordered) was covered. Description lists are another type of list provided by HTML. 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. The following example demonstrates a set of terms and definitions:

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

Description lists use a different wrapper than the other list types. This wrapper is represented with <dl>. In addition, wach term is then wrapped in a <dt> (description term) element, and each description is wrapped in a <dd> (description definition) element. The following is a marked up example:

<dl>

<dt>soliloquy</dt>

<dd>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.)</dd>

<dt>monologue</dt>

<dd>In drama, where a character speaks their thoughts out loud to share them with the audience and any other characters present.</dd>

<dt>aside</dt>

<dd>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.</dd>

</dl>

The browser default styles will display description lists with the descriptions indented somewhat from the terms. MDN’s styles follow this convention closely but emboldens the terms for extra definition. For example:

**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 character’s 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:**

**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.)

## Quotations

HTML also has features available for marking up quotation; which elements you use depends on whether you are marking up a clock 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 <em>HTML Block Quotation Element</em>) indicates that the enclosed text is an extended quotation.</p>

To turn this into a block quote, the following markup can be used:

<blockquote cite="https://developer.mozilla.org/en-US/docs/Web/HTML/Element/blockquote">

<p>The <strong>HTML <code>&lt;blockquote&gt;</code> Element</strong> (or <em>HTML Block

Quotation Element</em>) indicates that the enclosed text is an extended quotation.</p>

</blockquote>

Browser default styling will render this as an indented paragraph, as an indicator that it is a quote; MDN does this, but also adds extra styling as shown below:

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

### Inline Quotations

The content of the citations works in exactly the same way, except 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://developer.mozilla.org/en-US/docs/Web/HTML/Element/q">intended 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, however, browsers and screenreaders will often not handle any of the information associated with the citation. There is no way to get the browser to display the contents of cite without writing a solution using Javascript or CSS. In order to make the source of the quotation available on the page, the developer needs to make it available in the text via a link or some other appropriate way.

There is a <cite> element, however 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 a developer couldn’t link the text inside the <cite> to the quote source in come way. For example:

<p>According to the <a href="https://developer.mozilla.org/en-US/docs/Web/HTML/Element/blockquote">

<cite>MDN blockquote page</cite></a>:

</p>

<blockquote cite="https://developer.mozilla.org/en-US/docs/Web/HTML/Element/blockquote">

<p>The <strong>HTML <code>&lt;blockquote&gt;</code> Element</strong> (or <em>HTML Block

Quotation Element</em>) indicates that the enclosed text is an extended quotation.</p>

</blockquote>

<p>The quote element — <code>&lt;q&gt;</code> — is <q cite="https://developer.mozilla.org/en-US/docs/Web/HTML/Element/q">intended

for short quotations that don't require paragraph breaks.</q> -- <a href="https://developer.mozilla.org/en-US/docs/Web/HTML/Element/q">

<cite>MDN q page</cite></a>.</p>

Citations are styled in italic font by default.

## Abbreviations

Another fairly common element used 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 attribte.). The following are examples:

<p>We use <abbr title="Hypertext Markup Language">HTML</abbr> to structure our web documents.</p>

<p>I think <abbr title="Reverend">Rev.</abbr> Green did it in the kitchen with the chainsaw.</p>

When implemented, the description of the abbreviation stored in the “title” attribute will be displayed as a tooltip upon the user hovering over it.

**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. It is recommended to use <abbr> instead.**

## Marking Up Contact Details

HTML has an element for marking up contact details - <address>. This simply wraps around the contact details of an individual. For example:

<address>

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

</address>

The <address> section also offers the potential to include more complex markup and more detailed contact details such as the following 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>

## Superscript and Subscript

Occasionally, web content may need to be displayed in superscript or subscript, for example, when marking up items such as dates. Chemical formulae and mathematical equations. This ensures they will maintain the correct meaning. The <sup> and <sub> elements handle the formatting of text in this manner. 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>O<sub>2</sub>.</p>

<p>If x<sup>2</sup> is 9, x must equal 3 or -3.</p>

## Representing Computer Code

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

* <code>: For marking up generic pieces of 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 in 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 the 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.

The following markup contains examples of the above elements:

<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;</code> and <code>&lt;center&gt;</code>.</p>

<p>In the above JavaScript example, <var>para</var> represents a paragraph element.</p>

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

<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></pre>

## 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>

Obviously, humans write times and dates down using a variety of different formats, however, these forms are often not easily recognised by computers. If a developer wants to automatically grab the dates of all the events stored on a page and insert them into a calendar, there must be a way for the developer to extract datetime values in a suitable, unambiguous, machine readable format.

The <time> element allows this to happen by providing a machine readable datetime attribute alongside the human readable text representation.

The following example demonstrates the implementation of the <time> element in markup:

<!-- 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 in France</time>

<!-- Calling out a specific week number-->

<time datetime="2016-W04">The fourth week of 2016</time>

# Document and Website Structure

In addition to defining individual parts of the 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”). The following section looks into how to plan a basic website structure, and write the HTML used to represent this structure.

## Basic Sections of a Document

Webpages can, and will, look 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 art project, or 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 a website leads 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 centre 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, so on so forth… This component of the website will often vary in terms of content and style between different pages.

**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 (Search Engine Optimisation) purposes by providing links for quick access to popular content.

A typical website should be roughly structured something like this:

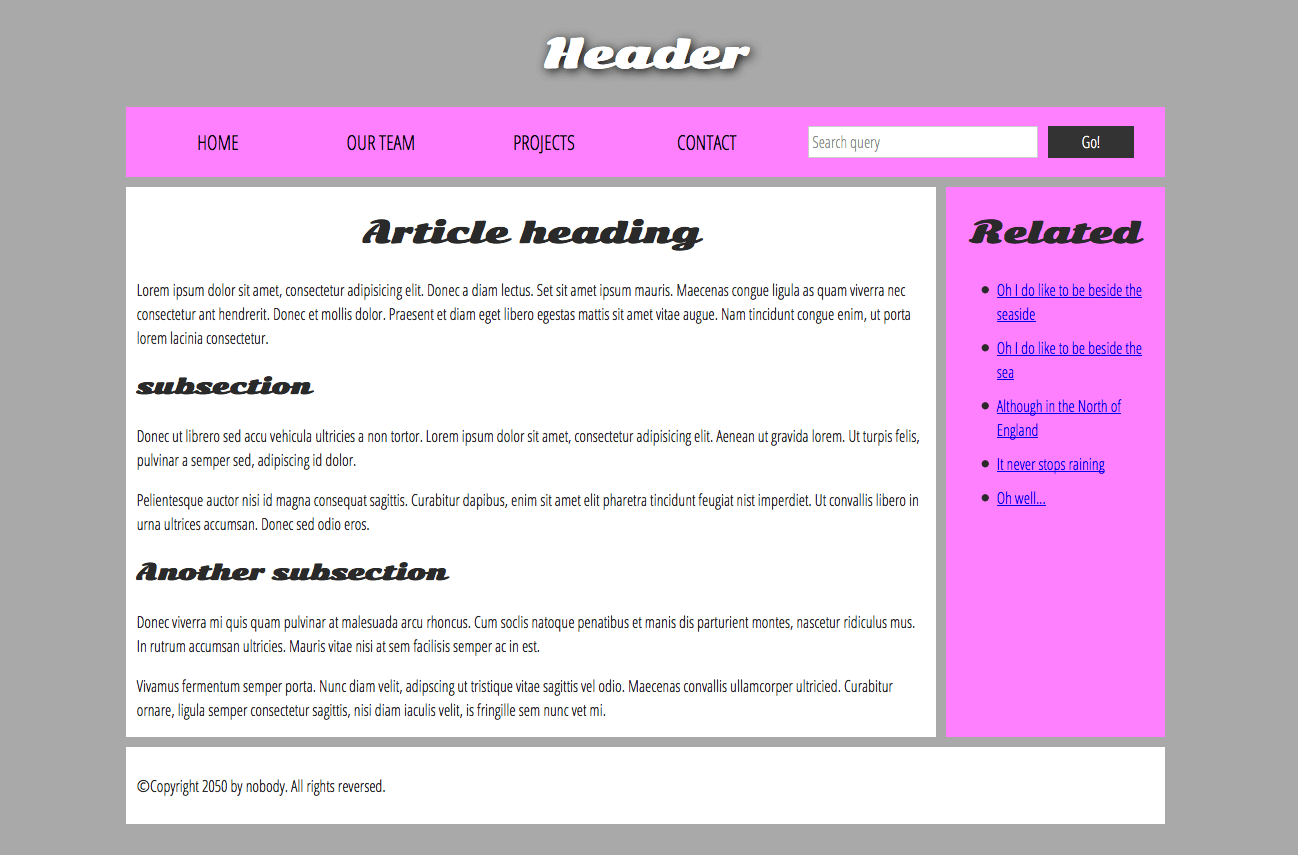


Figure 4- Demonstration of how a webpage should implement the key structural components (Image Sourced From: <https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction_to_HTML/Document_and_website_structure>)

## HTML For Structuring Content

The example shown in the previous section isn’t the most eloquent example of a web page, however, it is perfectly fine for illustrating a typical website layout. Some webpages have more complicated designs, with more columns, sections and other design additions.

With the right CSS, a developer can get their website looking however they want, using any of the HTML elements to wrap around the various sections discussed above, however as mentioned previously, semantics matter when using HTML. As such, the developer should always aim to use the right tool for the right job.

This is because visuals don’t always tell the whole story. A developer may well use colour and font to draw sighted user’s attention to the most useful parts of the content, such as the navigation and other related hyperlinks, but what about the visually impaired for example, who may not use concepts such as “pink” and “large font” in their understanding of a webpage.

It’s worth noting that colourblind people represent around 4% of the world’s population or, to put it in another way, approximately 1 in every 12 men and 1 in every woman are colour-blind. Blind and visually impaired people represent roughly 4-5% of the world’s population (in 2012 there were 285 million such people in the world, while the total population was around 7 billion.

In HTML code, you can markup sections of content based on their functionality – A developer 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 mentioned previously, there are consequences from not using the right element structure and semantics for the right job.

To implement such mark-up, HTML provides dedicated tags that can be used 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>.

## HTML Layout Elements in More Detail

It’s advisable to learn the overall meaning of all the HTML sectioning elements in detail – this is something that comes naturally as developer becomes more experienced in web development. A lot more detail can be found in the MDN HTML Element Reference (Found here: <https://developer.mozilla.org/en-US/docs/Web/HTML/Element>). For now, however, these are the main definitions to learn:

* **<main>**: Is common for content unique to the opened page. Use <main> only *once* per page, and put it directly inside <body>. Ideally this should be nested within other elements.
* **<article>**: Encloses a block of repeated content that makes sense on its own without the rest of the page (e.g. a single blog post).
* **<section>**: Is built for grouping together a single part of the page that constitutes one part 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 article’s can be broken up into different <section>’s, or <sections>’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 (this shouldn’t be confused with titles and headings).
* **<nav>**: Contains the specific main navigation functionality of the page. Secondary links would not go in the navigation wrapper.
* **Footer**: <footer> represents a group of end content for a page.

### 6.3.1. Non-Semantic Wrappers

Sometimes you’ll come across a situation where the ideal semantic element to group some items together or wrap some content. Sometimes a developer might want to 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. These should be used preferably with a suitable class attribute, to provide some kind of label for them so they can be easily targeted as a group.

<span> is an inline non-semantic element, which should only be used if a better, more semantic, alternative element can’t be found, or in instances where the developer doesn’t want to add any specific meaning. For example:

<p>The King walked drunkenly back to his room at 01:00, the beer doing nothing to aid

him as he staggered through the door <span class="editor-note">[Editor's note: At this point in the

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 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 screen reader users in finding it.

**WARNINGL: Div’s are so convenient to use that it’s easy to use them too much. They carry no semantic value and end up cluttering markup. Only use them when there is no better semantic solution and try to reduce their usage to the minimum otherwise, otherwise maintaining and updating HTML will become a lot harder.**

## 6.3.2. Line Breaks and Horizontal Rules.

Two elements occasionally used to format HTML are <br> and <hr>:

<br> creates a line break inside a paragraph it is the only way to enforce a rigid structure in a situation where the developer wants a series of 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 as one long line. With the <br> elements however, a new line is created after the appearance of a <br> elements in the markup. 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, this is displayed as a horizontal line. For example:

<p>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.</p>

<hr>

<p>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.</p>

## Planning A Simple Website

Once the structure of a simple webpage has been planned out, the next logical step is to try and work out what content is to be put on the whole website, what pages are needed, and how they should all be arranged and link to one another for the best experience. This is called **Information Architecture**. In a large complex website, a lot of planning can go into this process.

1. Some elements will be common to most, if not all, pages. Most commonly, these will often be the nav menu, header, and the footer. For a business site, it’s often wise to have the contact information on each page. The first step is to note down what is going to be common for every page.
2. Next, draw a rough sketch of what you might want the structure of each page to look like. The designer should not what each block is going to contain.
3. The next stage is to brainstorm all the other (not common to every page) content that is to be included on the website. This should be written down as a big list.
4. The next stage is to sort each piece of content that is to be included on the website that has just been written down in the big list into related groups in terms of what the content is and what it does. This naturally informs the designer of what parts might live together on different pages. This is a very similar technique to something known as “card sorting”.
5. Once the developer has grouped the content, it is then time to sketch a rough sitemap. A bubble should be used to represent each page and a line should be drawn to show the typical workflows between pages. The homepage will probably be in the centre, and link to most if now 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.

# Debugging HTML

The following section details the steps to take when HTML goes awry, and some of the tools that can be used to find and fix errors in HTML.

## HTML And Debugging

HTML is not compiled into an intermediate language before being parsed by the browser and outputs. HTML is interpreted much like Python or Java, and offers one a significantly easier syntax to learn than most other programming languages. This means HTML is a lot more permissive in terms of how it handles errors than a lot of other programming languages. This however, can be both good and bad in terms of debugging and writing stable markup.

### Permissive Code

When writing permissive code there are two main types of error:

* **Syntax Errors**: These are spelling errors in your code that cause the program to not run. These are easy to fix providing the developer has a familiarity with the syntax of the language and the error messages presented with the language.
* **Logic Errors**: These are errors where thet syntax is correct, but the code is not what the developer intended it to be, meaning the program runs incorrectly. These are often harder to fix than syntax errors, as there isn’t an error message to direct you to th 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 the user expected. This can, of course, 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 not have been as popular as it is today if the language had been stricter from the very beginning.**

The DOM inspector in most browsers can be used to debug code by analysing the changes the browser has made to the source code in it’s attempt to fix any errors.

### HTML Validation

The best strategy to check the validity of HTML documents is to run the page through the W3C Markup Validation Service. The web page will take the HTML document as an input, goes through it, and outputs a report that highlights and issues with the HTML.