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 2](#_Toc36496927)

[1.1. What Is HTML? 2](#_Toc36496928)

[1.2. Anatomy of a HTML Element. 2](#_Toc36496929)

[1.2.1. Nesting Elements 3](#_Toc36496930)

[1.2.2. Block Versus Inline Elements 3](#_Toc36496931)

[1.2.3. Empty Elements 5](#_Toc36496932)

[1.3. Attributes 5](#_Toc36496933)

[1.3.1. Adding Attributes to An Element 6](#_Toc36496934)

[1.3.2. Boolean Attributes 6](#_Toc36496935)

[1.3.3. Omitting Quotes Around Attribute Values 7](#_Toc36496936)

[1.3.4. Single or Double Quotes? 7](#_Toc36496937)

[1.4. Anatomy of an HTML Document 8](#_Toc36496938)

[1.4.1. Adding Some Features to an HTML Document 9](#_Toc36496939)

[1.4.2. Whitespace in HTML 11](#_Toc36496940)

[1.5. Entity Reference: Including Special Characters in HTML 11](#_Toc36496941)

[1.6. HTML Comments 12](#_Toc36496942)

[2. What’s in The Head? Metadata in HTML 13](#_Toc36496943)

[2.1. What is the HTML Head 13](#_Toc36496944)

[2.2. Adding a Title 14](#_Toc36496945)

[2.3. Metadata: The <meta> Element 14](#_Toc36496946)

[2.3.1. Specifying the Document’s Character Encoding 14](#_Toc36496947)

[2.3.2. Adding an Author and Description 15](#_Toc36496948)

[2.3.3. Other Types of Metadata 15](#_Toc36496949)

[2.4. Adding Custom Icons to Your Site 16](#_Toc36496950)

[2.5. Applying CSS and JavaScript to HTML 17](#_Toc36496951)

[2.6. Setting the Primary Language of the Document 18](#_Toc36496952)

[3. HTML Text Fundamentals 19](#_Toc36496953)

[3.1. The Basics: Headings and Paragraphs 19](#_Toc36496954)

[3.1.1. Implementing Structural Hierarchy 19](#_Toc36496955)

[3.1.2. Why Do We Need Structure? 21](#_Toc36496956)

[3.1.3. Why Do We Need Semantics? 22](#_Toc36496957)

[3.2. Lists 23](#_Toc36496958)

[3.2.1. Unordered Lists 23](#_Toc36496959)

[3.2.2. Ordered Lists 24](#_Toc36496960)

[3.2.3. Nesting Lists 25](#_Toc36496961)

[3.3. Emphasis and Importance 26](#_Toc36496962)

[3.3.1. Emphasis 26](#_Toc36496963)

[3.3.2. Strong Importance 26](#_Toc36496964)

[3.3.3. Italic, Bold and Underline 27](#_Toc36496965)

[4. Creating Hyperlinks 29](#_Toc36496966)

# 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.

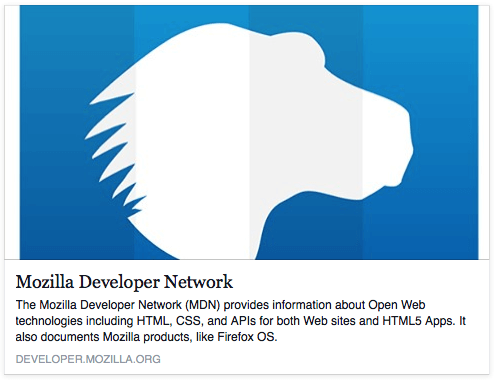
For example, Open Graph Data is a proprietary metadata protocol provided by Facebook that is designed to provide richer metadata for websites that can be used by Facebook when sharing links. For example:

<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.

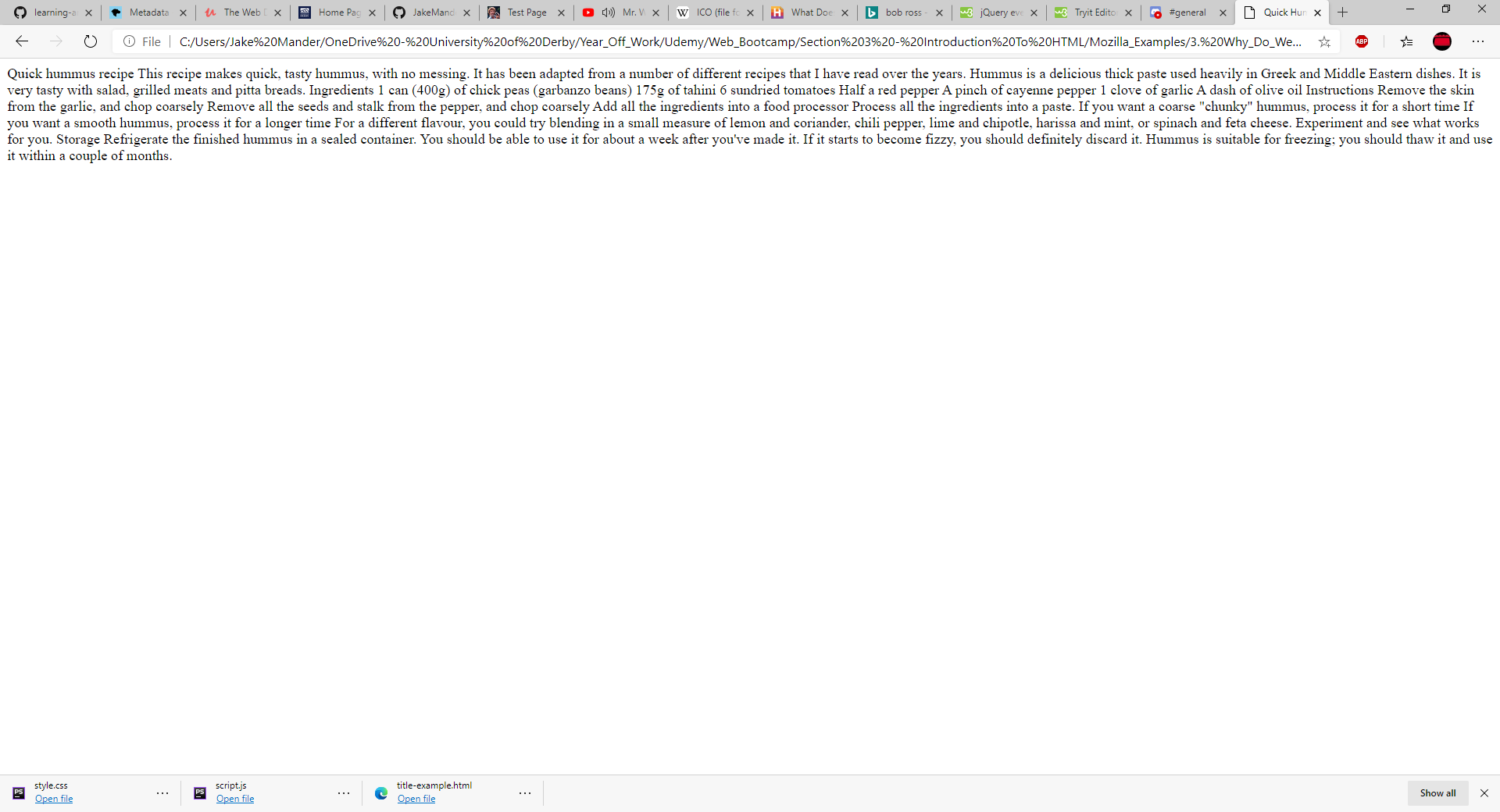
Storage

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.