HyperText Markup Language

Is a *markup language* that tells web browsers how to structure the web pages you visit. It can be as complicated or as simple as the web developer wants it to be. HTML consists of a series of elements, which you use to enclose, wrap, or *mark up* different parts of content to make it appear or act in a certain way. The enclosing tags can make content into a hyperlink to connect to another page, italicize words, and so on. For example, consider the following line of text:

My cat is very grumpy

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

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

The anatomy of our element is:

- **The opening tag:** This consists of the name of the element (in this example, *p* for paragraph), wrapped in opening and closing angle brackets. This opening tag marks where the element begins or starts to take effect. In this example, it precedes the start of the paragraph text.
- **The content:** This is the content of the element. In this example, it is the paragraph text.
- **The closing tag:** This is the same as the opening tag, except that it includes a forward slash before the element name. This marks where the element ends. Failing to include a closing tag is a common beginner error that can produce peculiar results.

The element is the opening tag, followed by content, followed by the closing tag.

Active learning: creating your first HTML element

<em> This is my text. </em>

Doing this should give the line **italic** text formatting! See your changes update live in the *Output* area.

Nesting elements

Elements can be placed within other elements. This is called *nesting*. If we wanted to state that our cat is very grumpy, we could wrap the word *very* in a <strong> element, which means that the word is to have strong(er) text formatting:

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

My cat is **very** grumpy.

There is a right and wrong way to do nesting. In the example above, we opened the p element first, then opened the strong element. For proper nesting, we should close the strong element first, before closing the p.

The following is an example of the *wrong* way to do nesting:

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

Block versus inline elements

There are two important categories of elements to know in HTML: block-level elements and inline elements.

- Block-level elements form a visible block on a page. A block-level element appears on a new line following the content that precedes it. Any content that follows a block-level element also appears on a new line. Block-level elements are **usually structural elements** on the page. For example, a block-level element might represent headings, paragraphs, lists, navigation menus, or footers. A block-level element wouldn't be nested inside an inline element, but it might be nested inside another block-level element.
- Inline elements are contained within block-level elements, and surround only small parts of the document's content (not entire paragraphs or groupings of content). An inline element will not cause a new line to appear in the document. It is typically used with text, for example an <a> element creates a hyperlink, and elements such as <em> or <strong> create emphasis.

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

  <p>fourth</p>

  <p>fifth</p>

  <p>sixth</p>
  ```

Void elements

Not all elements follow the pattern of an opening tag, content, and a closing tag. Some elements consist of a single tag, which is typically used to insert/embed something in the document. Such elements are called void elements. For example, the <img> element embeds an image file onto a page:

```
<img
  src="https://raw.githubusercontent.com/mdn/beginner-html-site/gh-pages/images/firefox-icon.png" alt="Firefox icon" />
```

Attributes

Elements can also have attributes. Attributes look like this:

attributes contain extra information about the element that won't appear in the content. In this example, the **class** attribute is an identifying name used to target the element with style information.

An attribute should have:

- A space between it and the element name. (For an element with more than one attribute, the attributes should be separated by spaces too.)
- The attribute name, followed by an equal sign.
- An attribute value, wrapped with opening and closing quote marks.

Active learning: Adding attributes to an element

Another example of an element is <a>. This stands for *anchor*. An anchor can make the text it encloses into a hyperlink. Anchors can take a number of attributes, but several are as follows:

href

> This attribute's value specifies the web address for the link. For example: href="https://www.mozilla.org/".

title

> The title attribute specifies extra information about the link, such as a description of the page that is being linked to. For example, title="The Mozilla homepage". This appears as a tooltip when a cursor hovers over the element.

target

> The target attribute specifies the browsing context used to display the link. For example, target="_blank" will display the link in a new tab. If you want to display the linked content in the current tab, just omit this attribute.

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

1. Add the <a> element.
2. Add the href attribute and the title attribute.
3. Specify the target attribute to open the link in the new tab.

```
<p>A link to my <a href="https://www.mozilla.org/" title="The Mozilla homepage"
target="_blank">favorite website</a>.</p>
```

Boolean attributes

Sometimes you will see attributes written without values. This is entirely acceptable. These are called Boolean attributes. Boolean attributes can only have one value, which is generally the same as the attribute name. For example, consider the disabled attribute, which you can assign to form input elements. (You use this to *disable* the form input elements so the user can't make entries. The disabled elements typically have a grayed-out appearance.) For example:

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

Omitting quotes around attribute values

If you look at code for a lot of other sites, you might come across a number of strange markup styles, including attribute values without quotes. This is permitted in certain circumstances, but it can also break your markup in other circumstances. For example, if we revisit our link example from earlier, we could write a basic version with *only* the href attribute, like this:

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

This is not good if we have multiple attributes.

Single or double quotes?

In this article you will also notice that the attributes are wrapped in double quotes. However, you might see single quotes in some HTML code. This is a matter of style. You can feel free to choose which one you prefer. Both of these lines are equivalent:

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

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

Make sure you don't mix single quotes and double quotes

To use quote marks inside other quote marks of the same type

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

Literal character	**Character reference equivalent**
<	&lt;
>	&gt;
"	&quot;
'	&apos;
&	&amp;

Anatomy of an HTML document

Individual HTML elements aren't very useful on their own. Next, let's examine how individual elements combine to form an entire HTML page:

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

Here we have:

1. <!DOCTYPE html>: The doctype. When HTML was young (1991-1992), doctypes were meant to act as links to a set of rules that the HTML page had to follow to be considered good HTML. Doctypes used to look something like this:

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

   More recently, the doctype is a historical artifact that needs to be included for everything else to work right. <!DOCTYPE html> is the shortest string of characters that counts as a valid doctype. That is all you need to know!

2. <html></html>: The <html> element. This element wraps all the content on the page. It is sometimes known as the root element.
3. <head></head>: The <head> element. This element acts as a container for everything you want to include on the HTML page, **that isn't the content** the page will show to viewers. This includes keywords and a page description that would appear in search results, CSS to style content, character set declarations, and more. You will learn more about this in the next article of the series.
4. <meta charset="utf-8">: The <meta> element. This element represents metadata that cannot be represented by other HTML meta-related elements, like <base>, <link>, <script>, <style> or <title>. The charset attributes set the character set for your document to UTF-8, which includes most characters from the vast majority of human written languages. With this setting, the page can now handle any textual content it might contain. There is no reason not to set this, and it can help avoid some problems later.
5. <title></title>: The <title> element. This sets the title of the page, which is the title that appears in the browser tab the page is loaded in. The page title is also used to describe the page when it is bookmarked.
6. <body></body>: The <body> element. This contains *all* the content that displays on the page, including text, images, videos, games, playable audio tracks, or whatever else.

Active learning: Adding some features to an HTML document

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

Whitespace in HTML

In the examples above, you may have noticed that a lot of whitespace is included in the code. This is optional. These two code snippets are equivalent:

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

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

No matter how much whitespace you use inside HTML element content (which can include one or more space character, but also line breaks), the HTML parser reduces each sequence of whitespace to a single space when rendering the code. So why use so much whitespace? The answer is readability.

It can be easier to understand what is going on in your code if you have it nicely formatted.

```
<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 has a mechanism to write comments in the code. Browsers ignore comments, effectively making comments invisible to the user.

To write an HTML comment, wrap it in the special markers

<!-- and -->. For example : <!-- <p>I am!</p> -->

What's in the head? Metadata in HTML

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

If you set your character encoding to ISO-8859-1, Japanish will not be displayed

Applying CSS and JavaScript to HTML

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

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

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

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

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

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

Active learning: applying CSS and JavaScript to a page

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

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