Getting started with HTML

In this article we cover the absolute basics of HTML, to get you started. We define elements, attributes, and all the other important terms you may have heard, and where they fit into the language. We also show how an HTML element is structured, how a typical HTML page is structured, and explain other important basic language features. Along the way, we'll play with some HTML to get you interested!

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

What is HTML?

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

```
My cat is very grumpy
```

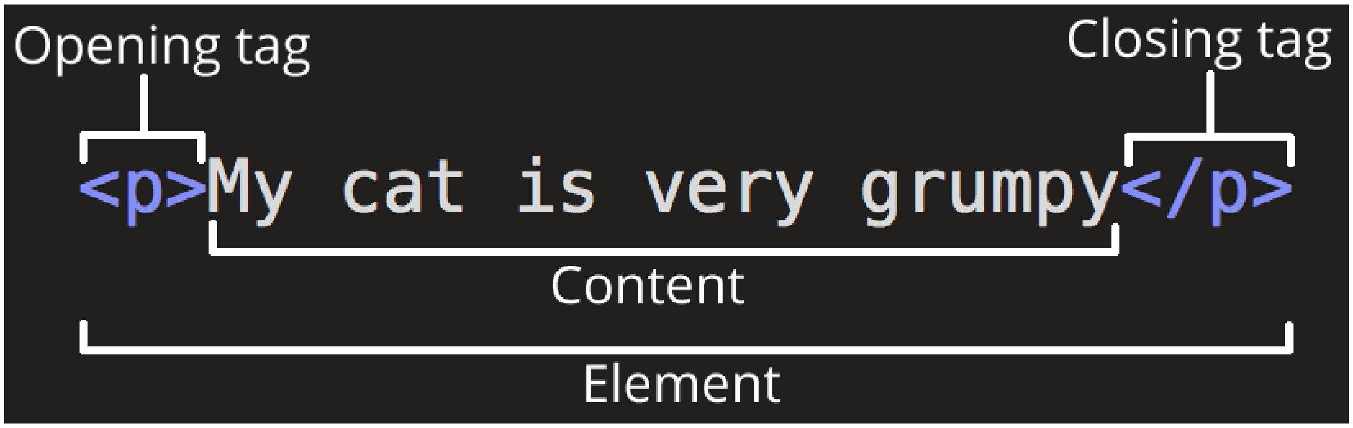
If we wanted the line to stand by itself, we could specify that it is a paragraph by enclosing it in a paragraph (`<p>`) element:

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

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

Anatomy of an HTML element

Let's explore our paragraph element a bit further:



The main parts of our element are:

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

Active learning: creating your first HTML element

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

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

Live output

This is my text.

Editable code

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

```
This is my text.
```

Reset Show solution

Nesting elements

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

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

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

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

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

Block versus inline elements

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

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

Take the following example:

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

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

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

```
firstsecondthird

fourth

fifth

sixth
```

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

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

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

Empty elements

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

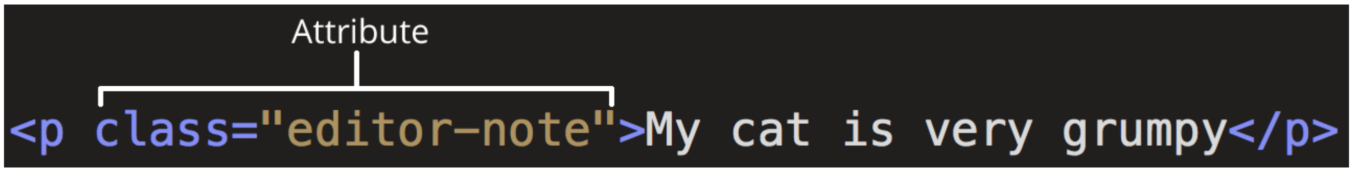
```
<img src="https://raw.githubusercontent.com/mdn/beginner-html-site/gh-pages/i
```

This would output the following on your page:

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

Attributes

Elements can also have attributes, which look like this:



```
<p class="editor-note">My cat is very grumpy</p>
```

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

An attribute should have:

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

Active learning: Adding attributes to an element

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

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

Edit the line below in the *Input* area to turn it into a link to your favorite website. First, add the `<a>` element. Second, add the `href` attribute and the `title` attribute. Lastly, specify the `target` attribute to open the link in the new tab. You'll be able to see your changes update live in the *Output* area. You should see a link that when hovered over displays the `title` attribute's content, and when clicked navigates to the web address in the `href` element. Remember that you need to include a space between the element name, and each attribute.

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

Live output

A link to my favorite website.

Editable code

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

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

Reset Show solution

Boolean attributes

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

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

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

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

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

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

Omitting quotes around attribute values

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

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

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

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

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

favorite website favorite website

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

Single or double quotes?

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

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

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

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

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

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

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

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

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

So you need to do this:

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

Anatomy of an HTML document

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

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

Here we have:

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

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

   However, these days no one really cares about them, and they are really just a historical artifact that needs to be included for everything to work right. `<!DOCTYPE html>` is the shortest string of characters that counts as a valid doctype; that's all you really need to know.
2. `<html></html>`: The `<html>` element. This element wraps all the content on the entire page, and is sometimes known as the root element.
3. `<head></head>`: The `<head>` element. This element acts as a container for all the stuff you want to include on the HTML page that *isn't* the content you are showing to your page's viewers. This includes things like keywords and a page description that you want to appear in search results, CSS to style our content, character set declarations, and more. You'll learn more about this in the next article in the series.
4. `<meta charset="utf-8">`: This element sets the character set your document should use to UTF-8, which includes most characters from the vast majority of human written

languages. Essentially it can now handle any textual content you might put on it. There is no reason not to set this, and it can help avoid some problems later.

5. `<title></title>`: The `<title>` element. This sets the title of your page, which is the title that appears in the browser tab the page is loaded in, and is used to describe the page when you bookmark/favorite it.
6. `<body></body>`: The `<body>` element. This contains *all* the content that you want to show to web users when they visit your page, whether that's text, images, videos, games, playable audio tracks, or whatever else.

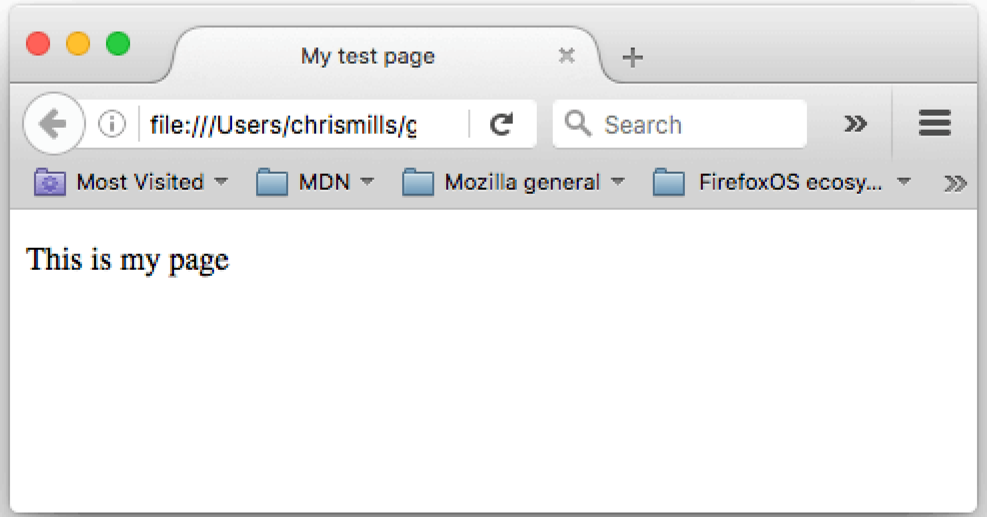
Active learning: Adding some features to an HTML document

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

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

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

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



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

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

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

Live output

This is my page

Editable code

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

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

Reset Show solution

Whitespace in HTML

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

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

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

No matter how much whitespace you use (which can include space characters, but also line breaks), the HTML parser reduces each one down to a single space when rendering the code. So why use so much whitespace? The answer is readability — it is so much easier to understand what is going on in your code if you have it nicely formatted, and not just bunched up together in a big mess. In our HTML we've got each nested element indented by two spaces more than the one it is sitting inside. It is up to you what style of formatting you use (how many spaces for each level of indentation, for example), but you should consider formatting it.

Entity references: Including special characters in HTML

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

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

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

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

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

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

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

In HTML, you define a paragraph using the

element.

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

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

HTML comments

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

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

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

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

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

I'm not inside a comment

Summary

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

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

See also

- Applying color to HTML elements using CSS

Overview: Introduction to HTML   Next