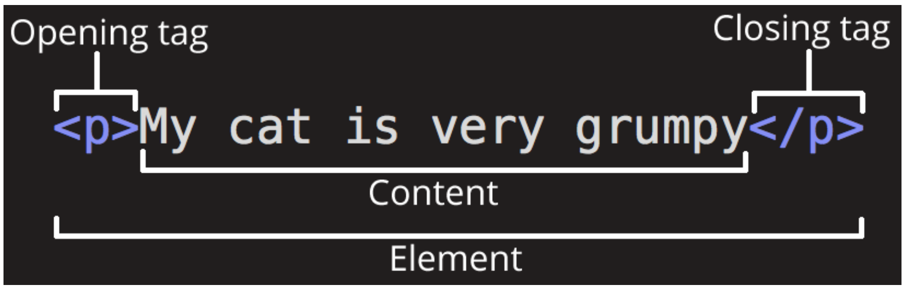
Introduction to HTML

HTML (Hypertext Markup Language) is not a programming language. It is a markup language that tells web browsers how to structure the web pages you visit. It also gives semantic meaning to contents of a webpage.



Nesting Elements

Elements can be placed within other elements. This is called nesting.

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

Block vs 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. A block-level element wouldn't be nested inside an inline element, but it might be nested inside another block-level element.

```
e.g: <h1>, <p>, <div>, <ul>
```

Inline elements are contained within block-level elements, and surround only small parts of the document’s content. An inline element will not cause a new line to appear in the document.

```
e.g: <a>, <em>, <strong>
```

Note: The terms block and inline, as used in this article, should not be confused with the types of CSS boxes that have the same names. While the names 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 reason HTML5 dropped these terms was to prevent this rather common confusion.

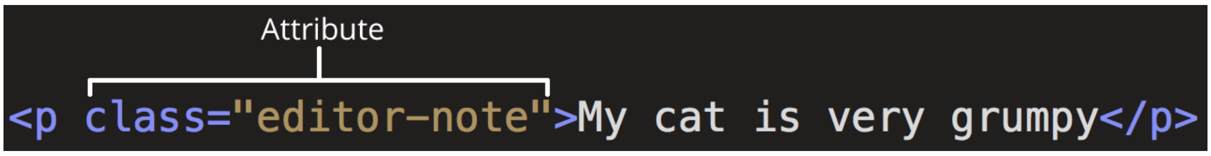
Empty Elements

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. 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">
```

Attributes

Elements can also have attributes. Attributes look like this:



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

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.

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, 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.)

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

Anatomy of an HTML document

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

1. <!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.
4. `<meta charset="utf-8">`: This element specifies the character set for your document to UTF-8, which includes most characters from the vast majority of human written languages. 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.

Whitespace in HTML

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. Both the below code snippets are equivalent.

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

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

Entity references

These are special codes that represent characters, to be used in these exact circumstances. Each character reference starts with an ampersand (&), and ends with a semicolon (;).

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

Note: You don't need to use entity references for any other symbols besides above, 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

Comments in HTML are written as below:

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

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

Head and Metadata

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.).

The HTML head is the contents of the <head> element — unlike the contents of the <body> element, the head's content is not displayed on the page. Instead, the head's job is to contain metadata about the document. The head also contains <title>, links to CSS and JS pages, <meta> tags of various kinds, among other things.
Many `<meta>` elements include `name` and `content` attributes:

Adding a Title

The <title> element is included in the head of the html document. The <title> element can be used to add a title to the document.

```
e.g <head>
       <title>My test page</title>
    </head>
```

If you try bookmarking the page you will see the <title> contents filled in as the suggested bookmark name.

Metadata: the <meta> element

Metadata is data that describes data, and HTML has an "official" way of adding metadata to a document — the <meta> element. There are a lot of different types of <meta> elements that can be included in your page's <head>. The most commonly used are shown as follows:

- **Specifying your document's character encoding**

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

This element specifies the document's character encoding — the character set that the document is permitted to use. utf-8 is a universal character set that includes pretty much any character from any human language. This means that your web page will be able to handle displaying any language; it's therefore a good idea to set this on every web page you create.

- **Adding an author and description**

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

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

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

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

<meta name="description" content="The MDN Web Docs Learning Area aims to 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 to be able to understand who wrote the page. Some

  content management systems have facilities to automatically extract page author information and make it available for such purposes.

  Specifying a description that includes keywords relating to the content of your page is useful as it has the potential to make your page appear higher in relevant searches performed in search engines (such activities are termed Search Engine Optimization, or SEO.)

- **Other types of metadata**

  A lot of the features you'll see on websites are proprietary creations, designed to provide certain sites (such as social networking sites) with specific pieces of information they can use.

  For example, **Open Graph Data** is a metadata protocol that Facebook invented to provide richer metadata for websites. In the MDN Web Docs sourcecode, you'll find below code. One effect of this is that when you link to MDN Web Docs on facebook, the link appears along with an image and description: a richer experience for users.

```
<meta property="og:image" content="https://developer.mozilla.org/static/img/opengraph-logo.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">
```

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

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

Adding custom icons to your site

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

The favicon is a 16-pixel square icon used in multiple places. You may see (depending on the browser) favicons displayed in the browser tab containing each open page, and next to bookmarked pages in the bookmarks panel.

A favicon can be added to your page by:

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

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

Applying CSS and JavaScript to HTML

CSS and JS 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.

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

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

Setting the primary language of the document

The primary language of your page be set by adding the lang attribute to the opening HTML tag as shown below:

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

Text Fundamentals

In programming, Semantics refers to the meaning of a piece of code — for example "what effect does running that line of JavaScript have?", or "what purpose or role does that HTML element have" (rather than "what does it look like?".)

In HTML, for example, the <h1> element is a semantic element, which gives the text it wraps around the role (or meaning) of "a top level heading on your page."

One of HTML's main jobs is to give text structure and meaning (also known as semantics) so that a browser can display it correctly.

The basics: headings and paragraphs

Most structured text consists of headings and paragraphs, whether you are reading a story, a newspaper, a college textbook, a magazine, etc. Structured content makes the reading experience easier and more enjoyable. In HTML, each paragraph has to be wrapped in a **<p>** element, like so:

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

Each heading has to be wrapped in a heading element:

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

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

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

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

Why do we need structure?

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

Why do we need semantics?

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

In a similar vein, we need to make sure we are using the correct elements, giving our content the correct meaning, function, or appearance. In this context, the `<h1>` element is also a semantic element, which gives the text it wraps around the role (or meaning) of "a top level heading on your page."

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

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

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

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

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

Lists

There are three different types of lists in HTML:

Unordered

Unordered lists are used to mark up lists of items for which the order of the items doesn't matter. Every unordered list starts off with a <ul> element—this wraps around all the list items:

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

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

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

Ordered

Ordered lists are lists in which the order of the items does matter.

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

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

Nesting Lists

It is perfectly ok to nest one list inside another one. You might want to have some sub-bullets sitting below a top-level bullet.

```
<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

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

Emphasis

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

I am glad you weren't late.

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

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

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

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

Strong importance

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

This liquid is **highly toxic**.

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

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

```
<p>This liquid is <strong>highly toxic</strong>.</p>
<p>I am counting on you. <strong>Do not</strong> be late!</p>
```

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

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

Italic, bold, underline...

The elements we've discussed so far have clearcut associated semantics. The situation with <b>, <i>, and <u> is somewhat more complicated. They came about so people could write bold, italics, or underlined text in an era when CSS was still supported poorly or not at all. Elements like this, which only affect presentation and not semantics, are known as **presentational elements** and should no longer be used because, as we've seen before, semantics is so important to accessibility, SEO, etc.

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

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

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

```
<!-- scientific names -->
<p>
  The Ruby-throated Hummingbird (<i>Archilochus colubris</i>)
  is the most common hummingbird in Eastern North America.
</p><!-- foreign words -->
<p>
  The menu was a sea of exotic words like <i lang="uk-latn">vatrushka</i>,
  <i lang="id">nasi goreng</i> and <i lang="fr">soupe à l'oignon</i>.
</p><!-- a known misspelling -->
<p>
  Someday I'll learn how to <u style="text-decoration-line: underline; 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 allow us to link documents to other documents or resources, link to specific parts of documents, or make apps available at a web address. Almost any web content can be converted to a link so that when clicked or otherwise activated the web browser goes to another web address (URL).

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

Anatomy of a link

A basic link is created by wrapping the text or other content, inside an `<a>` element and using the `href` attribute, also known as a Hypertext Reference, or target, that contains the web address

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

Adding supporting information with the title attribute

Another attribute you may want to add to your links is title. The title contains additional information about the link, such as which kind of information the page contains, or things to be aware of on the web site. Hovering over the link displays the title as a tooltip.

```
<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 contribut
e">the Mozilla homepage</a>.
</p>
```

Block level links

Almost any content can be made into a link, even block-level elements. For example, If you have an image you want to make into a link, use the `<a>` element and reference the image file with the `<img>` element.

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

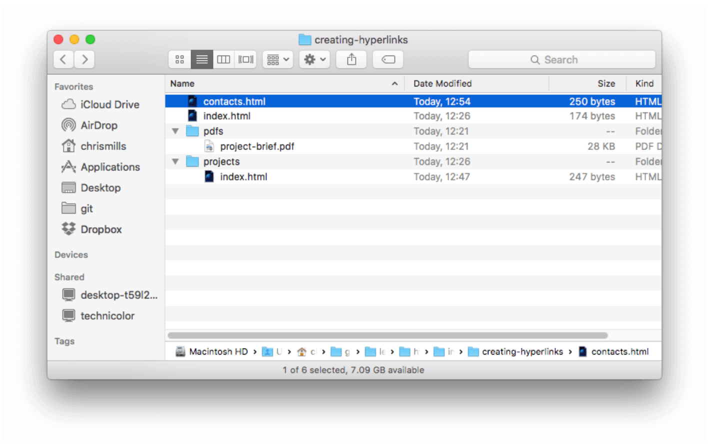
A quick primer on URLs and paths

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

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

URLs use paths to find files. Paths specify where the file you're interested in is located in the filesystem.

Let's look at an example of a directory structure, see the below **creating-hyperlinks** directory.



Name	Date Modified	Size	Kind
contacts.html	Today, 12:54	250 bytes	HTML
index.html	Today, 12:26	174 bytes	HTML
pdfs	Today, 12:21	--	Folde
project-brief.pdf	Today, 12:21	28 KB	PDF D
projects	Today, 12:26	--	Folde
index.html	Today, 12:47	247 bytes	HTML

Macintosh HD › U › c › g › le › h › ir › creating-hyperlinks › contacts.html

1 of 6 selected, 7.09 GB available

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

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

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

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

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

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

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

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

Note: You can combine multiple instances of these features into complex URLs, if needed, for example: ../../../complex/path/to/my/file.html.

Document fragments

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

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

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

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

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

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

Absolute versus relative URLs

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

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

relative URL: Points to a location that is relative to the file you are linking from, more like what we looked at in the previous section. For example, if we wanted to link from our example file at https://www.example.com/projects/index.html to a PDF file in the same directory, the URL would just be the filename — 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 https://www.example.com/projects/pdfs/project-brief.pdf.)

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

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

Link best practices

1. **Use clear link wording**

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

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

   Let's look at a specific example:

   Good *link text:* Download Firefox

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

Bad link text: Click here to download Firefox

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

Other tips:

- Don't repeat the URL as part of the link text.
- Don't say "link" or "links to" in the link text — it's just noise
- Keep your link text as short as possible — this is helpful because screen readers need to interpret the entire link text.
- Minimize instances where multiple copies of the same text are linked to different places. This can cause problems for screen reader users, if there's a list of links out of context that are labeled "click here", "click here", "click here".

2. **Use relative links wherever possible**

   you might think that it's a good idea to just use absolute links all the time because they don't break when a page is moved like relative links. However, you should use relative links wherever possible when linking to other locations within the *same website*. When you link to another website, you'll need to use an absolute link.

   - For a start, it's easier to scan your code — relative URLs are generally shorter than absolute URLs, which makes reading code much easier.
   - Second, it's more efficient to use relative URLs wherever possible. When you use an absolute URL, the browser starts by looking up the real location of the server on the Domain Name System (DNS) — Then it goes to that server and finds the file that's being requested. With a relative URL, the browser just looks up the file that's being requested on the same server. If you use absolute URLs where relative URLs would do, you're constantly making your browser do extra work, meaning that it will perform less efficiently.

3. **Linking to non-HTML resources — leave clear signposts**

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

   For example:

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

```
<p><a href="https://www.example.com/large-report.pdf">
  Download the sales report (PDF, 10MB)
</a></p>
<p><a href="https://www.example.com/video-stream/" target="_blank">
  Watch the video (stream opens in separate tab, HD quality)
</a></p>
<p><a href="https://www.example.com/car-game">
  Play the car game (requires Flash)
</a></p>
```

4. **Use the download attribute when linking to a download**

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

```
<a href="https://download.mozilla.org/?product=firefox-latest-ssl&os=win64&lang=en-US"download="firefox-latest-64bit-installer.
exe">
  Download Latest Firefox for Windows (64-bit) (English, US)
</a>
```

E-mail links

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

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

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

In fact, the email address is optional. If you omit it and your href is "mailto:", a new outgoing email window will be opened by the user's email client with no destination address. This is often useful as "Share" links that users can click to send an email to an address of their choosing.

Specifying e-mail details

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

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

```
<a href="mailto:nowhere@mozilla.org?cc=name2@rapidtables.com&bcc=name3@rapidtables.com&subject=The%20subject%20of%20the%20email&bod
y=The%20body%20of%20the%20email">
  Send mail with cc, bcc, subject and body
</a>
```

Note the use of the question mark (?) to separate the main URL from the field values, and ampersands (&) to separate each field in the mailto: URL. This is standard URL query notation.

Advanced Text Formatting

Description lists

The purpose of these lists is to mark up a set of items and their associated descriptions, such as terms and definitions, or questions and answers. Description lists use a different wrapper than the other list types — <`dl>`; in addition each term is wrapped in a `<dt>` (description term) element, and each description is wrapped in a `<dd>` (description definition) element.

```
<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
  <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,
</dl>
```

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

```
<dl>
  <dt>aside</dt>
  <dd>In drama, where a character shares a comment only with the audience for humorous or dramatic effect. This is usually a feelin
g, thought, or piece of additional background information.
  </dd>
  <dd>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.)
  </dd>
</dl>
```

Quotations

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

Blockquotes

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

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

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

```
<p>Here below is a blockquote...</p>
<blockquote cite="https://developer.mozilla.org/en-US/docs/Web/HTML/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; the paragraph above the quotation is there to demonstrate that.

Inline quotations

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

```
<p>The quote element — <code>&lt;q&gt;</code> — is <q cite="https://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:

Citations

The content of the cite attribute sounds useful, but unfortunately browsers, screenreaders, etc. don't really do much with it. There is no way to get the browser to display the contents of cite, without writing your own solution using JavaScript or CSS. If you want to make the source of the quotation available on the page you need to make it available in the text via a link or some other appropriate way.

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

```
<p>According to the <a href="/en-US/docs/Web/HTML/Element/blockquote"><cite>MDN blockquote page</cite></a>:
</p>
```

Abbreviations

The abbreviation tag `<abbr>` — this is used to wrap around an abbreviation or acronym, and provide a full expansion of the term (included inside a `title` attribute.) Let's look at a couple of examples:

```
<p>We use <abbr title="Hypertext Markup Language">HTML</abbr> to structure our web documents.</p>
<p>I think <abbr title="Reverend">Rev.</abbr> Green did it in the kitchen with the chainsaw.</p>
```

Marking up contact details

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

```
<address>
  <p>
    Chris Mills<br>
    Manchester<br>
    The Grim North<br>
    UK
  </p>
    <ul>
      <li>Tel: 01234 567 890</li>
      <li>Email: me@grim-north.co.uk</li>
    </ul>
</address>
```

Hyperlinks can also be contained inside `<address>` tags

Superscript and subscript

The `<sup>` and `<sub>` elements handle this job.

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

```
<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>
```

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

```
<!-- Standard simple date -->
<time datetime="2016-01-20">20 January 2016</time>
<!-- Just year and month -->
<time datetime="2016-01">January 2016</time>
<!-- Just month and day -->
<time datetime="01-20">20 January</time>
<!-- Just time, hours and minutes -->
<time datetime="19:30">19:30</time>
<!-- You can do seconds and milliseconds too! -->
<time datetime="19:30:01.856">19:30:01.856</time>
<!-- Date and time -->
<time datetime="2016-01-20T19:30">7.30pm, 20 January 2016</time>
<!-- Date and time with timezone offset -->
<time datetime="2016-01-20T19:30+01:00">7.30pm, 20 January 2016 is 8.30pm 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 your page (such as "a paragraph" or "an image"), HTML also boasts a number of block level elements used to define areas of your website (such as "the header", "the navigation menu", "the main content column").

Basic sections of a typical webpage

Webpages can and will look pretty different from one another, but they all tend to share similar standard components:

header:

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

navigation bar:

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

main content:

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

sidebar:

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

footer:

A strip across the bottom of the page that generally contains fine print, copyright notices, or contact info. It's a place to put common information (like the header) but usually, that information is not critical or secondary to the website itself.

The footer is also sometimes used for SEO purposes, by providing links for quick access to popular content.

HTML for structuring content

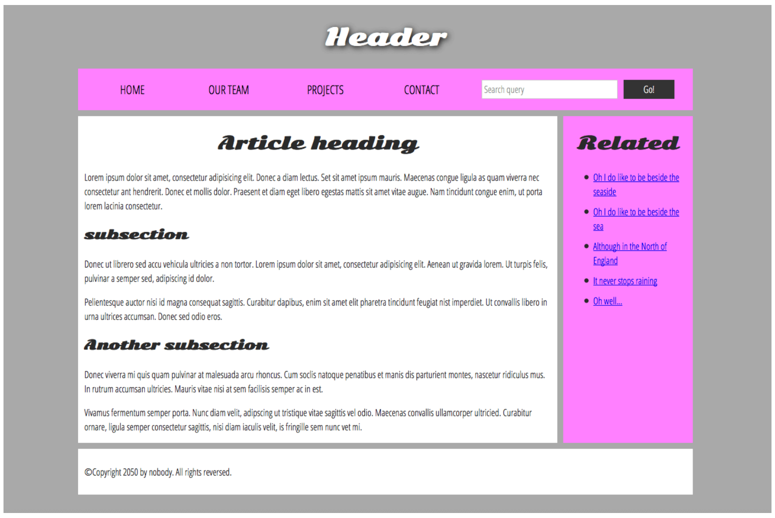
Some websites have more columns, some are a lot more complex, but you get the idea. With the right CSS, you could use pretty much any elements to wrap around the different sections and get it looking how you wanted, but as discussed before, we need to respect semantics and **use the right element for the right job.** This is because visuals don't tell the whole story. We use color and font size to draw sighted users' attention to the most useful parts of the content, like the navigation menu and related links, but what about visually impaired people for example, who might not find concepts like "pink" and "large font" very useful?

In your HTML code, you can mark up sections of content based on their functionality — you can use elements that represent the sections of content described above unambiguously, and assistive technologies like screenreaders can recognize those elements and help with tasks like "find the main navigation", or "find the main content." To implement such semantic mark up, HTML provides dedicated tags that you can use to represent such sections, for example:

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

Below is a typical webpage with the necessary semantic and structure tags used to group the page contents into various components.

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

    <title>My page title</title>
    <link href="https://fonts.googleapis.com/css?family=Open+Sans+Condensed:300|Sonsie+One" rel="stylesheet" type="text/css">
    <link rel="stylesheet" href="style.css">

    <!-- the below three lines are a fix to get HTML5 semantic elements working in old versions of Internet Explorer-->
    <!--[if lt IE 9]>
      <script src="https://cdnjs.cloudflare.com/ajax/libs/html5shiv/3.7.3/html5shiv.js"></script>
    <![endif]-->
  </head>

  <body>
    <!-- Here is our main header that is used across all the pages of our website -->

    <header>
      <h1>Header</h1>
    </header>

    <nav>
      <ul>
        <li><a href="#">Home</a></li>
        <li><a href="#">Our team</a></li>
        <li><a href="#">Projects</a></li>
        <li><a href="#">Contact</a></li>
      </ul>

       <!-- A Search form is another common non-linear way to navigate through a website. -->

       <form>
         <input type="search" name="q" placeholder="Search query">
         <input type="submit" value="Go!">
       </form>
     </nav>

    <!-- Here is our page's main content -->
    <main>

      <!-- It contains an article -->
      <article>
        <h2>Article heading</h2>

        <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Donec a diam lectus. Set sit amet ipsum mauris. Maecenas congue l

        <h3>Subsection</h3>
        <p>Donec ut librero sed accu vehicula ultricies a non tortor. Lorem ipsum dolor sit amet, consectetur adipisicing elit. Aenean

        <p>Pelientesque auctor nisi id magna consequat sagittis. Curabitur dapibus, enim sit amet elit pharetra tincidunt feugiat nist

        <h3>Another subsection</h3>

        <p>Donec viverra mi quis quam pulvinar at malesuada arcu rhoncus. Cum soclis natoque penatibus et manis dis parturient montes,

        <p>Vivamus fermentum semper porta. Nunc diam velit, adipscing ut tristique vitae sagittis vel odio. Maecenas convallis ullamco
      </article>

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

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

    </main>

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

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

  </body>
</html>
```


Header

HOME OUR TEAM PROJECTS CONTACT Search query Go!

Article heading

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Donec a diam lectus. Set sit amet ipsum mauris. Maecenas congue ligula as quam viverra nec consectetur ant hendrerit. Donec et mollis dolor. Praesent et diam eget libero egestas mattis sit amet vitae augue. Nam tincidunt congue enim, ut porta lorem lacinia consectetur.

subsection

Donec ut librero sed accu vehicula ultricies a non tortor. Lorem ipsum dolor sit amet, consectetur adipisicing elit. Aenean ut gravida lorem. Ut turpis felis, pulvinar a semper sed, adipiscing id dolor.

Pelientesque auctor nisi id magna consequat sagittis. Curabitur dapibus, enim sit amet elit pharetra tincidunt feugiat nist imperdiet. Ut convallis libero in urna ultrices accumsan. Donec sed odio eros.

Another subsection

Donec viverra mi quis quam pulvinar at malesuada arcu rhoncus. Cum soclis natoque penatibus et manis dis parturient montes, nascetur ridiculus mus. In rutrum accumsan ultricies. Mauris vitae nisi at sem facilisis semper ac in est.

Vivamus fermentum semper porta. Nunc diam velit, adipscing ut tristique vitae sagittis vel odio. Maecenas convallis ullamcorper ultricied. Curabitur ornare, ligula semper consectetur sagittis, nisi diam iaculis velit, is fringille sem nunc vet mi.

Related

- Oh I do like to be beside the seaside
- Oh I do like to be beside the sea
- Although in the North of England
- It never stops raining
- Oh well...

©Copyright 2050 by nobody. All rights reversed.

HTML layout elements in more detail

It's good to understand the overall meaning of all the HTML sectioning elements in detail

- `<main>` is for content unique to this page. Use `<main>` only once per page, and put it directly inside `<body>`. Ideally this shouldn't be nested within other elements.
- `<article>` encloses a block of related content that makes sense on its own without the rest of the page (e.g., a single blog post). It is a piece of information which is reusable and self-contained. A tweet is an excellent example of an `article` tag
- `<section>` is similar to `<article>`, but it is more for grouping together a single part of the page that constitutes one single piece of functionality (e.g., a mini map, or a set of article headlines and summaries), or a theme. It's considered best practice to begin each section with a heading; also note that you can break `<article>`s up into different `<section>`s, or `<section>`s up into different `<article>`s, depending on the context.

- `<aside>` contains content that is not directly related to the main content but can provide additional information indirectly related to it (glossary entries, author biography, related links, etc.).
- `<header>` represents a group of introductory content. If it is a child of `<body>` it defines the global header of a webpage, but if it's a child of an `<article>` or `<section>` it defines a specific header for that section (try not to confuse this with titles and headings).
- `<nav>` contains the main navigation functionality for the page. Secondary links, etc., would not go in the navigation.
- `<footer>` represents a group of end content for a page.

Non-semantic wrappers

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

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

```
<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>
```

`<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,

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

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

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

Line breaks and horizontal rules

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

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

Without the `<br>` elements, the paragraph would just be rendered in one long line since HTML ignores most whitespace.

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

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

1. Bear in mind that you'll have a few elements common to most (if not all) pages — such as the navigation menu, and the footer content.
2. Next, draw a rough sketch of what you might want the structure of each page to look like (it might look like our simple website above). Note what each block is going to be.
3. Now, brainstorm all the other (not common to every page) content you want to have on your website — write a big list down.
4. Next, try to sort all these content items into groups, to give you an idea of what parts might live together on different pages. This is very similar to a technique called Card sorting.
5. Now try to sketch a rough sitemap — have a bubble for each page on your site, and draw lines to show the typical workflow between pages. The homepage will probably be in the center, and link to most if not all of the others; most of the pages in a small site should be available from the main navigation, although there are exceptions. You might also want to include notes about how things might be presented.