HTML FAQ

**Basic Structure**

1. [How to create a basic HTML document](https://developer.mozilla.org/en-US/Learn/HTML/Introduction_to_HTML/Getting_started#Anatomy_of_an_HTML_document)

**Step 1:** Open Notepad (PC)

Windows 8 or later:

Open the Start Screen (the window symbol at the bottom left on your screen). Type Notepad.

Windows 7 or earlier:

Open Start > Programs > Accessories > Notepad

**Step 1:** Open TextEdit (Mac)

Open Finder > Applications > TextEdit

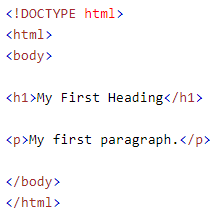
Also change some preferences to get the application to save files correctly. In Preferences > Format > choose "Plain Text"

Then under "Open and Save", check the box that says "Ignore rich text commands in HTML files".

Then open a new document to place the code.

**Step 2:** Write Some HTML

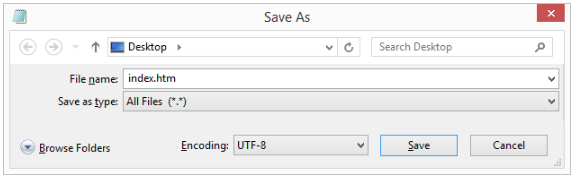
Write or copy some HTML into Notepad.



**Step 3:** Save the HTML Page

Save the file on your computer. Select File > Save as in the Notepad menu.

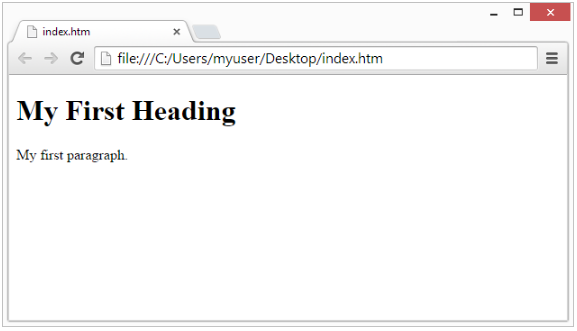
Name the file "index.htm" and set the encoding to UTF-8 (which is the preferred encoding for HTML files).



**Step 4:** View the HTML Page in Your Browser

Open the saved HTML file in your favorite browser (double click on the file, or right-click - and choose "Open with").

The result will look much like this:



1. [How to divide a webpage into logical sections](https://developer.mozilla.org/en-US/docs/Learn/HTML/Howto/Divide_a_webpage_into_logical_sections)

To implement such semantic mark up, HTML provides dedicated tags that you can use to represent such sections, for example:

**header:** <header>.

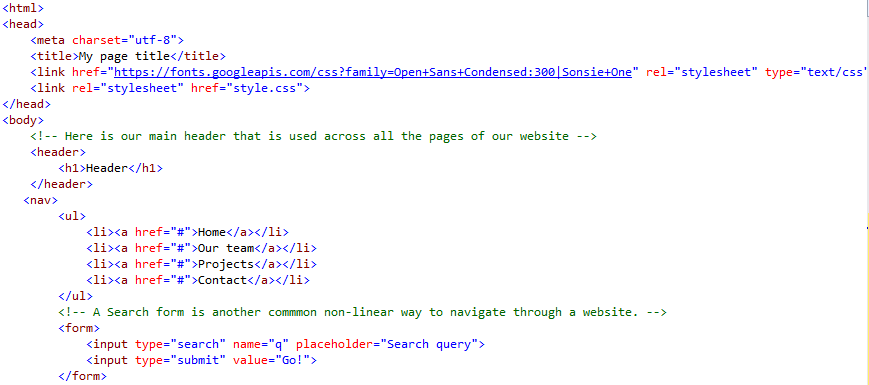
**navigation bar:** <nav>.

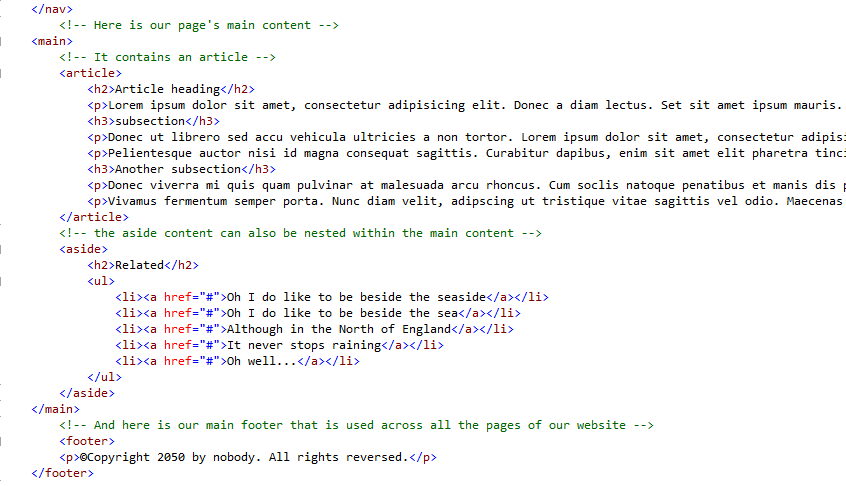
**main content:** <main>, with various content subsections represented by <article>, <section>, and <div> elements.

**sidebar:** <aside>; often placed inside <main>.

**footer:** <footer>.

**Example**





**HTML Layout elements**

* **<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.)
* **<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.) 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.

1. [How to set up a proper structure of headings and paragraphs](https://developer.mozilla.org/en-US/Learn/HTML/Introduction_to_HTML/HTML_text_fundamentals#The_basics_headings_and_paragraphs)

In HTML, each paragraph has to be wrapped in a <p> element, like so:



Each heading has to be wrapped in a heading element:



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.

**Basic text-level semantics**

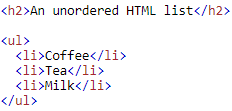
1. How to create list of items with HTML

**Unordered HTML List**

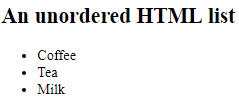
An unordered list starts with the <ul> tag. Each list item starts with the <li> tag.

The list items will be marked with bullets (small black circles) by default:

**Example**



**Output**

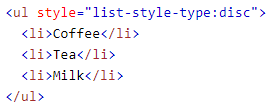


**Unordered HTML List - Choose List Item Marker**

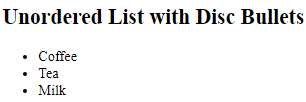
The CSS list-style-type property is used to define the style of the list item marker:

|  |  |
| --- | --- |
| Value | Description |
| disc | Sets the list item marker to a bullet (default) |
| circle | Sets the list item marker to a circle |
| square | Sets the list item marker to a square |
| none | The list items will not be marked |

**Example - Disc**



**Output:**

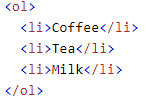


**Ordered HTML List**

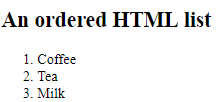
An ordered list starts with the <ol> tag. Each list item starts with the <li> tag.

The list items will be marked with numbers by default:

**Example**



**Output:**

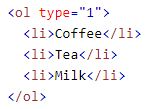
****

**Ordered HTML List - The Type Attribute**

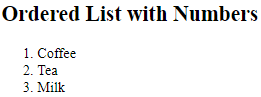
The type attribute of the <ol> tag, defines the type of the list item marker:

|  |  |
| --- | --- |
| Type | Description |
| type="1" | The list items will be numbered with numbers (default) |
| type="A" | The list items will be numbered with uppercase letters |
| type="a" | The list items will be numbered with lowercase letters |
| type="I" | The list items will be numbered with uppercase roman numbers |
| type="i" | The list items will be numbered with lowercase roman numbers |

**Example: Numbers**



**Output:**

****

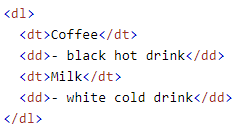
**HTML Description Lists**

HTML also supports description lists.

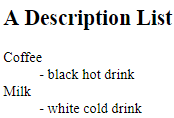
A description list is a list of terms, with a description of each term.

The <dl> tag defines the description list, the <dt> tag defines the term (name), and the <dd> tag describes each term:

**Example**



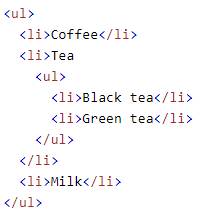
**Output**



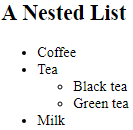
**Nested HTML Lists**

List can be nested (lists inside lists):

**Example**

****

**Output**

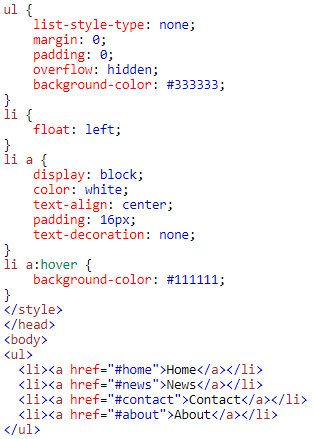
****

**Horizontal Lists**

HTML lists can be styled in many different ways with CSS.

One popular way is to style a list horizontally, to create a menu:

**Example**



**Output**



1. How to stress or emphasize content

The <em> tag is a phrase tag. It renders as emphasized text.

All phrase tags:

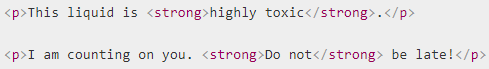
|  |  |
| --- | --- |
| Tag | Description |
| <em> | Renders as emphasized text |
| <strong> | Defines important text |
| <code> | Defines a piece of computer code |
| <samp> | Defines sample output from a computer program |
| <kbd> | Defines keyboard input |
| <var> | Defines a variable |

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. Browsers style this as bold text by default, but shouldn't use this tag purely to get bold styling. To do that, use a <span> element and some CSS, or perhaps a <b> element (see below.)

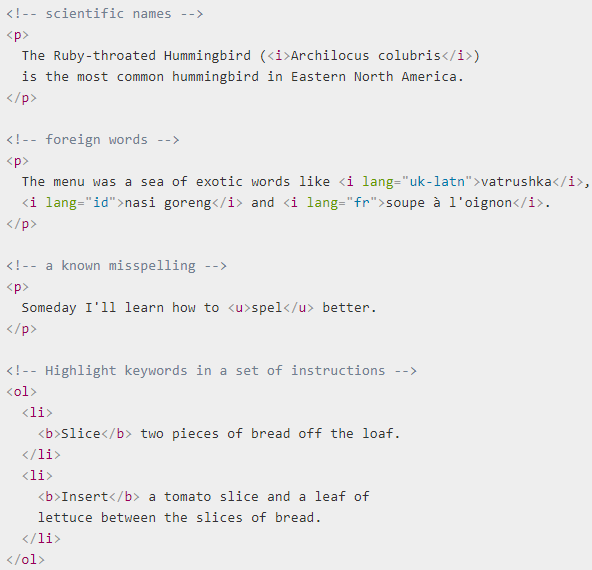


You can nest strong and emphasis inside one another if desired:



1. How to indicate that text is important

* <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...

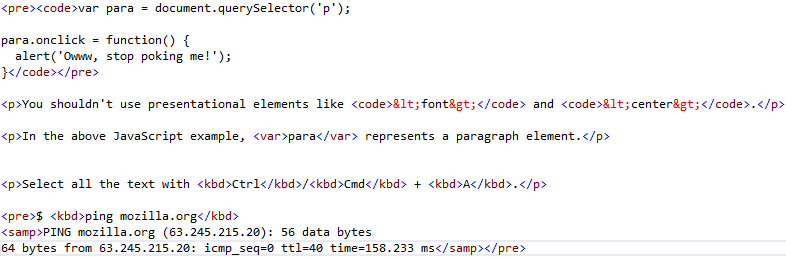


1. How to display computer code with HTML

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

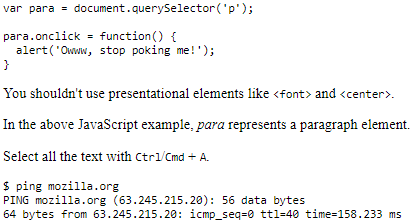
* <code>: For marking up generic pieces of computer code.
* <pre>: For retaining whitespace (generally code blocks) - if you indentation or excess whitespace inside text, browsers will ignore it and you will not see it on rendered page. If you wrap the text in <pre></pre> tags however, whitespace rendered 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.

**Example**

****

**Output:**

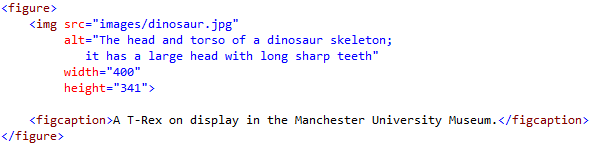
The above code will look like this



1. How to annotate images and graphics

HTML5 <figure> and <figcaption> elements is to provide a semantic container for figures, and clearly linking the figure to the caption.

**For Example**



The <figcaption> element tells browsers, and assistive technology, that the caption describes the other content of the <figure> element.

A figure doesn't have to be an image. It is an independent unit of content that:

* Expresses your meaning in a compact, easy-to-grasp way.
* Could go in several places in the page's linear flow.
* Provides essential information supporting the main text.

A figure could be several images, a code snippet, audio, video, equations, a table, or something else.

1. How to mark abbreviations and make them understandable

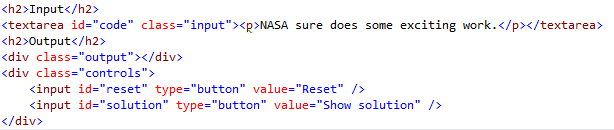
<abbr> - this is used to wrap around an abbreviation or acronym, and provide a full expansion of the term.

**Example**

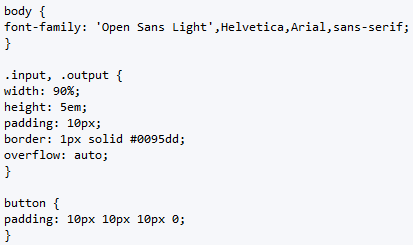


We will work on mark up an abbreviation.

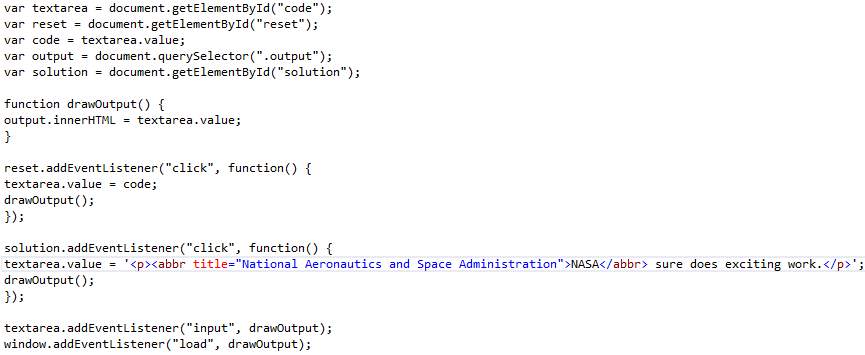
**HTML**

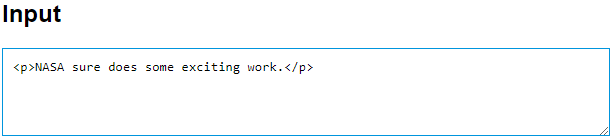


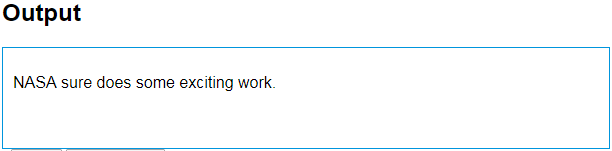
**CSS**



**JS**



****



1. How to add quotations and citations to WebPages

**Blockquotes**

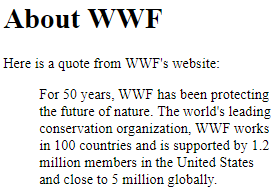
The <blockquote> tag specifies a section that is quoted from another source.

Browsers usually indent <blockquote> elements.

**Example**



**Output:**

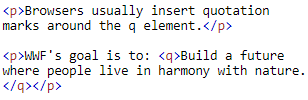


**Inline Quotations**

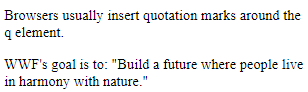
The HTML <q> element defines a short quotation.

Browsers usually insert quotation marks around the <q> element.

**Example**



**Output**



Browser default styling will render this as normal text put in quotes to indicate a quotation, like so:

The quote element - <q> - is "intended for short quotations that don't require paragraph breaks."

**Citations**

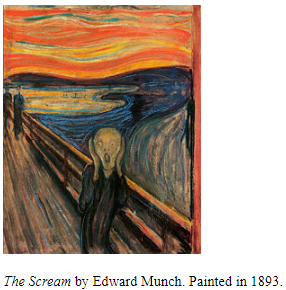
The <cite> tag defines the title of a work (e.g. a book, a song, a movie, a TV show, a painting, a sculpture, etc.).

**Note:** A person's name is not the title of a work.

**Example**

****

**Output:**

****

1. How to define terms with HTML

The <dfn> tag represents the defining instance of a term in HTML.

The defining instance is often the first use of a term in a document.

The nearest parent of the <dfn> tag must also contain the definition/explanation for the term inside <dfn>.

The term inside the <dfn> tag can be any of the following:

1. The content of the <dfn> element (without a title attribute):

**Example**

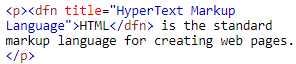
****

**Output:**

****

1. The title attribute of the <dfn> tag:

**Example**

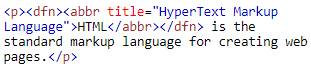
****

**Output**

****

1. The title attribute of an <abbr> tag inside the <dfn> element:

**Example**

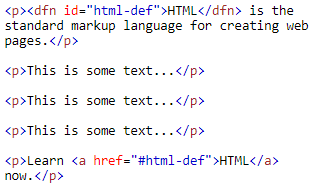
****

**Output**

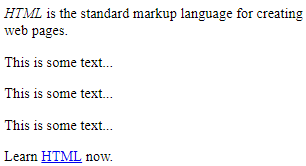
****

You can also add an id attribute to the <dfn> element. Then, whenever a term is used, it can refer back to the definition using an <a> tag:

**Example**

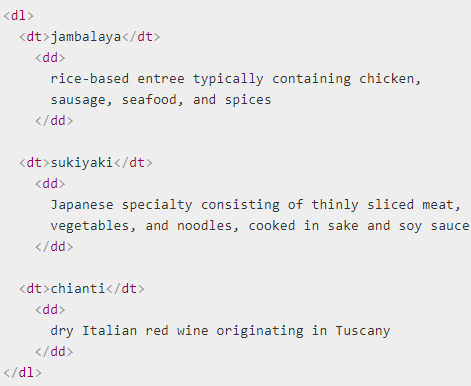
****

**Output**

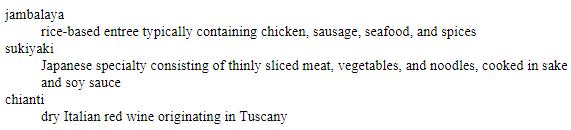
****

**A simple example**

Here's a simple example describing kinds of food and drink:

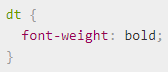


**Output:**

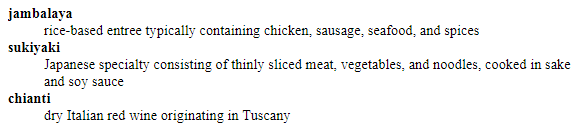
****

**Improving the visual output**

If you want the keywords to stand out better, you could try bolding them. Remember, HTML is not a visual medium; we need CSS for all visual effects. The CSS font-weight property is what you need here:

****

**Output:**

****

### Hyperlinks

1. How to create a hyperlink

A basic link is created by wrapping the text, you want to turn into a link inside an <a> element, and giving it an href attribute (also known as a Hypertext Reference , or target) that will contain the web address you want the link to point to.



**Output:**

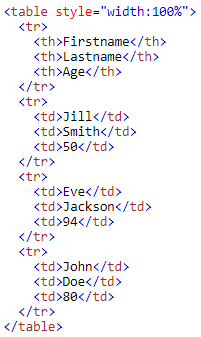
****

1. How to create a table of contents with HTML

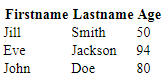
An HTML table is defined with the <table> tag.

Each table row is defined with the <tr> tag. A table header is defined with the <th> tag. By default, table headings are bold and centered. A table data/cell is defined with the <td> tag.

**Example**



**Output**

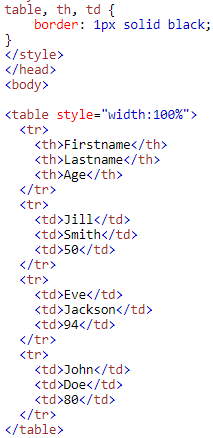


**HTML Table - Adding a Border**

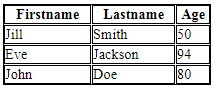
If you do not specify a border for the table, it will be displayed without borders.

A border is set using the CSS border property:

**Example**

****

**Output**

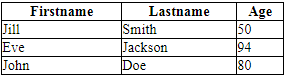
****

**HTML Table - Collapsed Borders**

If you want the borders to collapse into one border, add the CSS border-collapse property:



**Output**

****

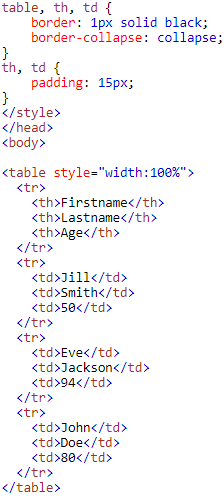
**HTML Table - Adding Cell Padding**

Cell padding specifies the space between the cell content and its borders.

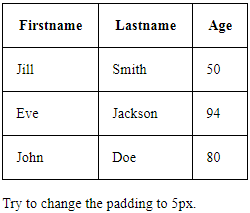
If you do not specify a padding, the table cells will be displayed without padding.

To set the padding, use the CSS padding property:

**Example**

****

**Output**

****

**HTML Table - Left -align Headings**

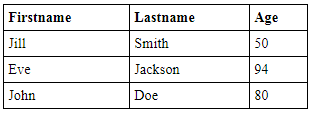
By default, table headings are bold and centered.

To left-align the table headings, use the CSS text-align property:

**Example**

****

**Output**

****

**HTML Table - Adding Border Spacing**

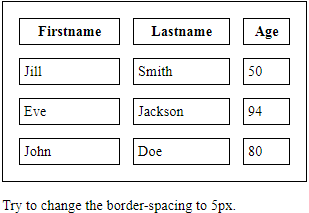
Border spacing specifies the space between the cells.

To set the border spacing for a table, use the CSS border-spacing property:

**Example**

****

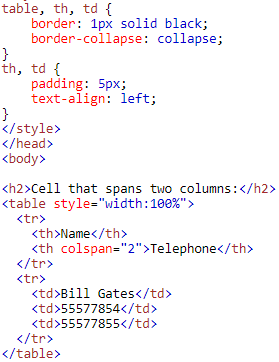
**Output**

****

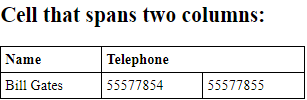
**HTML Table - Cells that Span Many Columns**

To make a cell span more than one column, use the **colspan** attribute:

**Example**

****

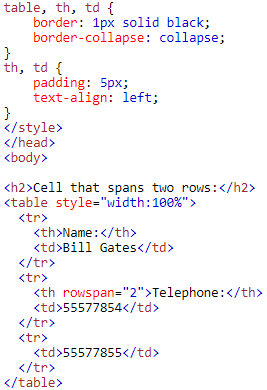
**Output**

****

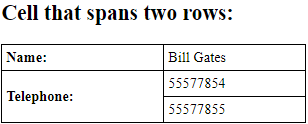
**HTML Table - Cells that Span Many Rows**

To make a cell span more than one row, use the **rowspan** attribute:

**Example**

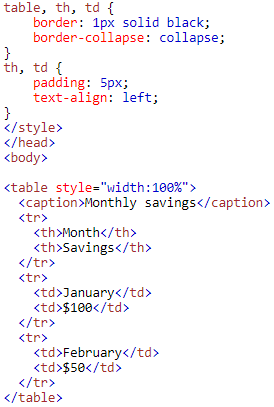


**Output**

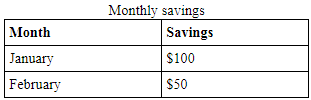


**HTML Table - Adding a Caption**

To add a caption to a table, use the **<caption>** tag:



**Output**



**Images & multimedia**

1. [How to add images to a webpage](https://developer.mozilla.org/en-US/Learn/HTML/Multimedia_and_embedding/Images_in_HTML#How_do_we_put_an_image_on_a_webpage)

In order to put a simple image on a webpage, we use the <img> element. This is an empty element (meaning that it has no text content or closing tag) that requires a minimum of one attribute to be useful.

The src attribute contains a path pointing to the image you want to embed in the page, which can be a relative or absolute URL, in the same way as <a> element href attribute values.

So for example, if your image is called dinosaur.jpg, and it sat in the same directory as your HTML page, you could embed the image like so:



If the image was in an images subdirectory, which was inside the same directory as the HTML page, then you'd embed it like this:



You could embed the image using its absolute URL, for example:



But this is pointless, as it just makes the browser do more work, looking up the IP address from the DNS server all over again, etc. You'll almost always keep the images for your web site on the same server as your HTML.

**Output:**



1. [How to add video content to a webpage](https://developer.mozilla.org/en-US/Learn/HTML/Multimedia_and_embedding/Video_and_audio_content)

The <video> element allows to embed a video easily. A simple example looks like this:



The features of note are:

**src**

In the same way as for the <img> element, the src (source) attribute contains a path to the video you want to embed.

**controls**

Users must be able to control video and audio playback (it's especially critical for people who have epilepsy.) You must either use the controls attribute to include the browser's own control interface, or build your interface using the appropriate JavaScript API. At minimum, the interface must include a way to start and stop the media, and to adjust the volume.

**The paragraph inside the <video> tags**

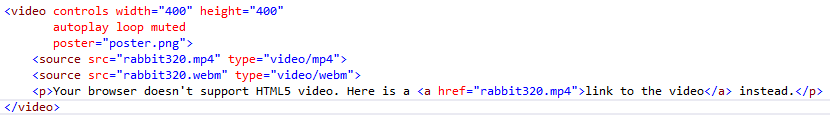
This is called fallback content — this will be displayed if the browser accessing the page doesn't support the <video> element, allowing us to provide a fallback for older browsers. This can be anything you like; in this case we've provided a direct link to the video file, so the user can at least access it some way regardless of what browser they are using.

**Output:**

****

**Other <video> features**

There are a number of other features can include on an HTML5 video. Take a look at an example, below:



**Output:**

****

The new features are:

**width and height**

You can control the video size either with these attributes or with CSS. In both cases, videos maintain their native width-height ratio — known as the aspect ratio. If the aspect ratio is not maintained by the sizes you set, the video will grow to fill the space horizontally, and the unfilled space will just be given a solid background color by default.

**autoplay**

This attribute makes the audio or video start playing right away while the rest of the page is loading. You are advised not to use autoplaying video (or audio) on your sites, because users can find it really annoying.

**loop**

This attribute makes the video (or audio) start playing again whenever it finishes. This can also be annoying, so only use if really necessary.

**muted**

This attribute causes the media to play with the sound turned off by default.

**poster**

This attribute takes as its value the URL of an image, which will be displayed before the video is played. It is intended to be used for a splash or advertising screen.

**preload**

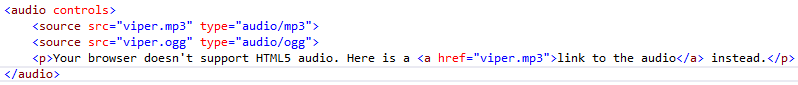
this attribute is used in the element for buffering large files. It can take one of 3 values:

* "none" does not buffer the file
* "auto" buffers the media file
* "metadata" buffers only the metadata for the file

1. [How to add audio content to a webpage](https://developer.mozilla.org/en-US/Learn/HTML/Multimedia_and_embedding/Video_and_audio_content)

The <audio> element works in exactly the same way as the <video> element, with a few small differences as outlined below.

**Example:**

****

**Output:**

****

* The <audio> element doesn't support the width/height attributes — again, there is no visual component, so there is nothing to assign a width or height to.
* It also doesn't support the poster attribute — again, no visual component.

Other than this, <audio> supports all the same features as <video>.

**Scripting & Styling**

1. [How to use CSS within a webpage](https://developer.mozilla.org/en-US/Learn/CSS/Introduction_to_CSS/How_CSS_works#How_to_apply_your_CSS_to_your_HTML)

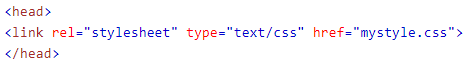
There are three ways of inserting a style sheet:

* External style sheet
* Internal style sheet
* Inline style

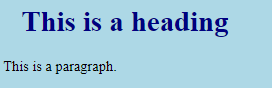
**External Style Sheet**

External Style Sheet can change the look of an entire website by changing just one file.

Each page must include a reference to the external style sheet file inside the <link> element. The <link > element goes inside the <head> section:

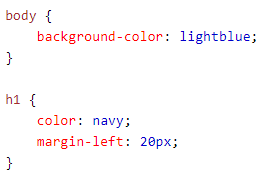


**Output:**



An external style sheet can be written in any editor. The file should not contain any html tags. The style sheet file must be saved with a .css extension.

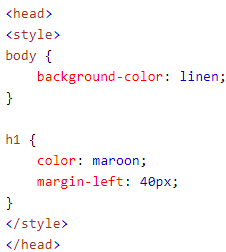
Here is how the "mystyle.css" looks:



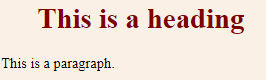
**Internal Style Sheet**

An internal style sheet may be used if one single page has a unique style.

Internal styles are defined within the <style> element, inside the <head> section of an HTML page:



**Output:**

****

**Inline Styles**

An inline style may be used to apply a unique style for a single element.

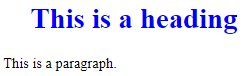
To use inline styles, add the style attribute to the relevant element. The style attribute can contain any CSS property.

The example below shows how to change the color and the left margin of a <h1> element:

**Example:**

****

**Output:**

****

1. [How to use JavaScript within a webpage](https://developer.mozilla.org/en-US/docs/Learn/HTML/Howto/Use_JavaScript_within_a_webpage)

Within a browser, JavaScript doesn't do anything by itself. You can run JavaScript from inside your HTML webpages. To call JavaScript code from within HTML, you need the <script> element.

There are two ways to use script, depending on whether linking to an external script or embedding a script right in webpage.

**Linking an external script**

External scripts are practical when the same code is used in many different web pages.

JavaScript files have the files extension **.js**

To use an external script, put the name of the script file in the src (source) attribute of a <script> tag:



You can place an external script reference in <head> or <body> as you like.

The script will behave as if it was located exactly where the <script> tag is located.

**External JavaScript Advantages**

Placing scripts in external files has advantages:

* It separates HTML and code.
* It makes HTML and JavaScript easier to read and maintain.
* Cached JavaScript files can speed up page loads

To add several script files to one page - use several script tags:

**Example**

****

**Embedded content**

1. [How to embed a webpage within another webpage](https://developer.mozilla.org/en-US/Learn/HTML/Multimedia_and_embedding/Other_embedding_technologies)

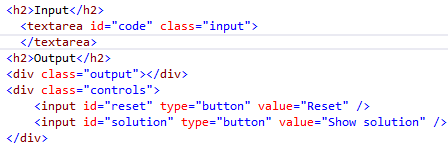
Let's look at how Youtube allows us to embed a video in any page we like using an **<iframe>.**

* First, go to Youtube and find a video you like.
* Below the video you'll find the share button — select this to display the sharing options.
* Select the *Embed* tab option and you'll be given some <iframe> code — copy this.
* Insert into the Input box below, and see what the result is in the Output.

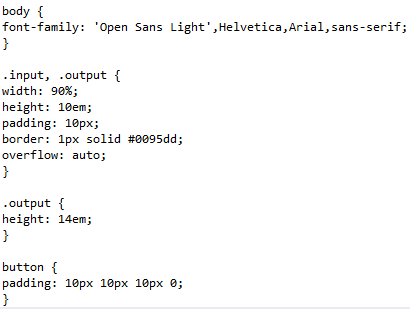
You could also try embedding a Google Map in the example:

* Go to Google Maps and find a map you like.
* Click on the "Hamburger Menu" (three horizontal lines) in the top left of the UI.
* Select the Share or embed map option.
* Select the Embed map option, which will give you some <iframe> code — copy this.
* Insert it into the Input box below, and see what the result in the Output.

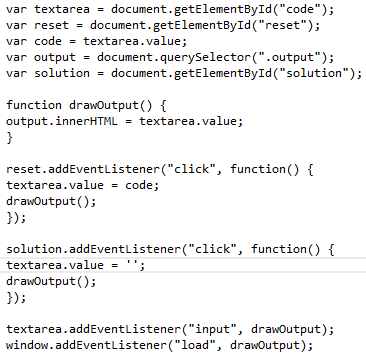
**HTML**

****

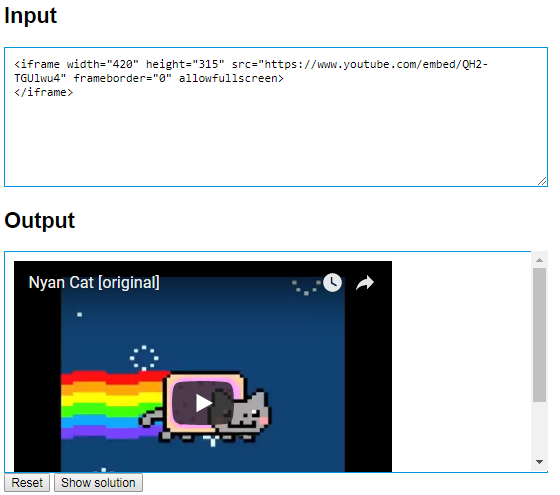
**CSS**

****

**JS**

****

**Output:**

****

1. [How to add Flash content within a webpage](https://developer.mozilla.org/en-US/Learn/HTML/Multimedia_and_embedding/Other_embedding_technologies#The_%3Cembed%3E_and_%3Cobject%3E_elements)

The purpose of a plug-in is to extend the functionality of a web browser.

Plug-ins are computer programs that extend the standard functionality of a web browser.

Example of well-known plug-ins are Java applets.

Plug-ins can be added to web pages with the <object> tag or the <embed> tag.

Plug-ins can be used for many purposes: display maps, scan for viruses, verfiy your bank id, etc.

**The <object> Element**

The <object > element is supported by all browsers.

The <object> element defines an embedded object within an HTML documnet.

It is used to embed plug-ins (like Java applets, PDF readers, Flash Players) in web pages.

**Example**



**Output**

****

The <object > element can also be used to include HTML in HTML:

**Example**



**Output**

****

Or images if you like



**Output**



**The <embed> element**

The <embed> element is supported in all major browsers.

The <embed> element also defines an embedded object within an HTML document.

Web browsers have supported the <embed> element for a long time. However, it has not been a part of the HTML specification before HTML5.

**Example**



**Output:**

****

The <embed> element can also be used to include HTML in HTML:

**Example**



**Output**

****

Or images if you like

**Example**

****

**Output**

****

**Uncommon or Advanced Problems**

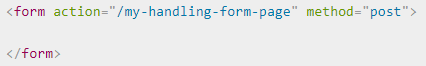
**Forms**

1. [How to create a simple Web form](https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Forms/My_first_HTML_form)

To create a contact form, we will use the following HTML elements: <form>, <label>, <input>, <textarea>, and <button>.

**The <form> element**

All HTML forms start with a <form> element like this:



* The action attribute defines the location (URL) where the form's collected data should be sent when it is submitted.
* The method attribute defines which HTTP method to send the data with (it can be "get" or "post").

For now, add the above <form> element into HTML body.

**The <label>, <input>, and <textarea> elements**

Our contact form is really simple and contains three text fields, each with a label. The input field for the name will be a basic single-line text field, the input field for the e-mail will be a single-line text field that accepts only an e-mail address, and the input field for the message will be a basic multiline text field.

Update your form code to look like the below.



The <div> elements are there to conveniently structure our code and make styling easier.

Note the use of the for attribute on all <label> elements; it's a formal way to link a label to a form widget.

This attribute references the id of the corresponding widget.

On the <input> element, the most important attribute is the type attribute. This attribute is extremely important because it defines the way the <input> element behaves. It can radically change the element so pay attention to it.

* In our simple example we use the value text for the first input — the default value for this attribute. It represents a basic single-line text field that accepts any kind of text input.
* For the second input, we use the value email that defines a single-line text field that only accepts a well-formed e-mail address. This last value turns a basic text field into a kind of "intelligent" field that will perform some checks on the data typed by the user.



On the contrary, if you want to define the default value of a <textarea>, just have to put that default value between the starting and ending tag of the <textarea> element, like this:



**The <button> element**

we just have to add a button to allow the user to send their data once they have filled out the form. This is simply done by using the <button> element; add the following just above the closing </form> tag:

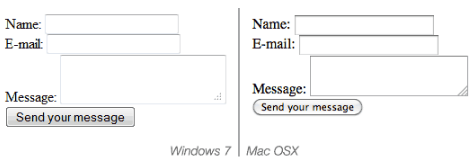


<button> element also accepts a type attribute — this accepts one of three values: submit, reset, or button.

* A click on a submit button (the default value) sends the form's data to the web page defined by the action attribute of the <form> element.
* A click on a reset button resets all the form widgets to their default value immediately. From a UX point of view, this is considered bad practice.
* A click on a button button does... nothing! That sounds silly, but it's amazingly useful for building custom buttons with JavaScript.

**Basic form styling**

You have finished writing form's HTML code, try saving it and looking at it in a browser.

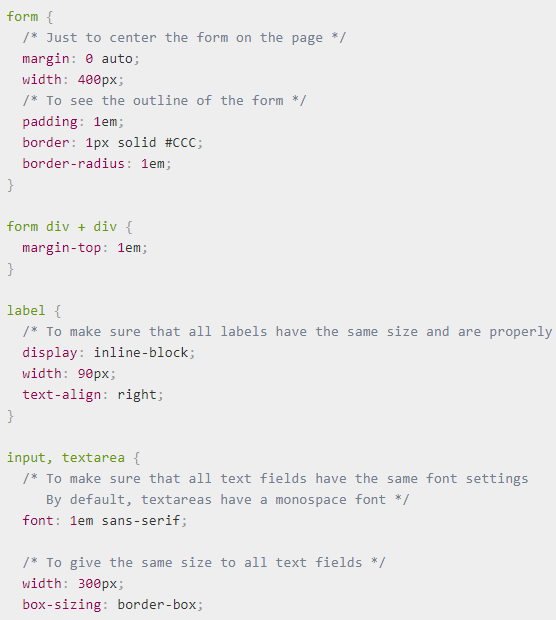


Add some CSS to make it look OK.

First of all, add a <style> element to page, inside HTML head. It should look like so:

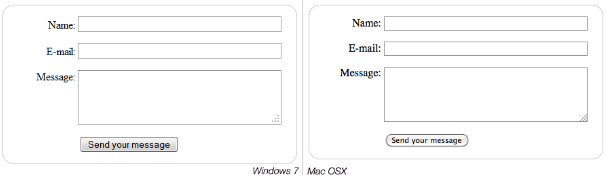


Inside the style tags, add the following CSS, just as shown:





**Output:**



1. [How to structure a Web form](https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Forms/How_to_structure_an_HTML_form)
   1. To start with, make a local copy of our blank template file and the CSS for our payment form in a new directory on your computer.
   2. First of all, apply the CSS to the HTML by adding the following line inside the HTML <head>:



* 1. Next, start your form off by adding the outer <form> element:



* 1. Inside the <form> tags, start by adding a heading and paragraph to inform users how required fields are marked:



* 1. Next add a larger section of code into the form, below our previous entry. Here we are wrapping the contact information fields inside a distinct <section> elements. Moreover, we have a set of two radio buttons, each of which we are putting inside its own list (<li>) element. Last, we have two standard text <input>s and their associated <label> elements, each contained inside a <p>, plus a password input for entering a password. Add this code to your form now:





1. Now we'll turn to the second <section> of our form - the payment information. Here we have three distinct widgets along with their labels, each contained inside a <p>. The first is a drop down menu (<select>) for selecting credit card type. the second is an <input> element of type number, for entering a credit card number. The last one is an <input> element of type date, for entering the expiration date of the card. Again, enter the following below the previous section:



1. The last section we'll add is alot simpler, containing only a <button> of type submit, for submitting the form data. Add this to the bottom of form.



**Output**

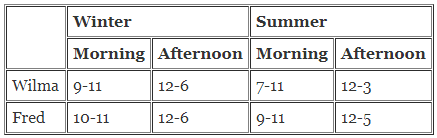


**Tabular Information**

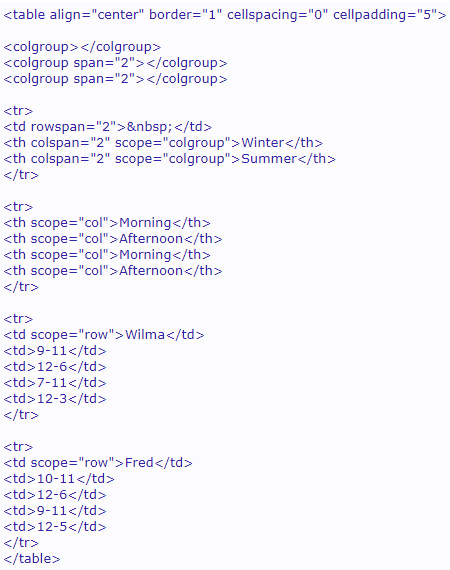
1. [How to create a data spreadsheet](https://developer.mozilla.org/en-US/docs/Learn/HTML/Howto/Create_a_data_spreadsheet)
2. [How to make HTML tables accessible](https://developer.mozilla.org/en-US/docs/Learn/HTML/Howto/Make_HTML_tables_accessible)

To create the table, we must set up the three column groups and set scope = "colgroup" on the headers that span two columns.

(using colspan):



The code would look like this:

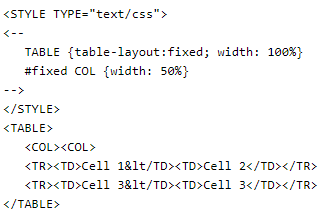


First, we have three <colgroup> tags, one group holding the data headers on the left (Fred and Wilma), the next holding two columns (headed by Winter), and the third group containing the last two columns (Summer). Under the main headings at the top we have further headings, whose scope is just the column underneath them.

So, if a capable browser looks at, say, the cell with 7-11 in it, it can tell the user that this cell pertains to Wilma and is headed by Summer and Morning.

1. [How to optimize HTML table rendering](https://developer.mozilla.org/en-US/docs/Learn/HTML/Howto/Optimize_HTML_table_rendering)

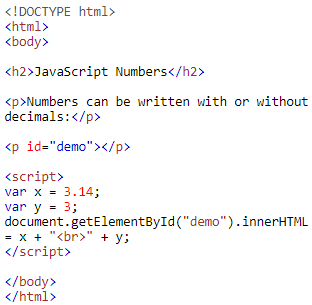
With browsers that support CSS2 like IE5.5 and NS6, tables can get a big speed boost with the new table-layout property. When you set the table-layout to a "fixed" table layout, you are fixing the column widths (and optionally the column heights) for the entire table. For these newer browsers this setting greatly increases the parsing and display performance of tables. This allows the table to be rendered progressively to the screen.

****

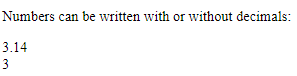
**Data Representation**

1. [How to represent numeric values with HTML](https://developer.mozilla.org/en-US/docs/Learn/HTMLHowto/Represent_numeric_values_with_HTML)

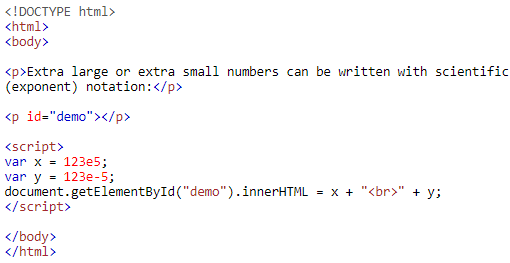
JavaScript has only one type of number. Numbers can be written with or without decimals.



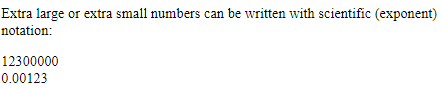
**Output**

****

Extra large or extra small numbers can be written with scientific (exponent) notation:



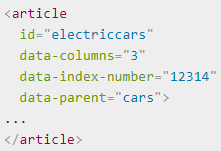
**Output**

****

1. [How to use data attributes](https://developer.mozilla.org/en-US/docs/Learn/HTML/Howto/Use_data_attributes)

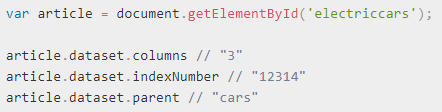
**HTML Syntax**

Any attribute on any element whose attribute name starts with data-is a data attribute. Say you have an article and you want to store some extra information that doesn’t have any visual representation. Just use data attributes for that:



**JavaScript access**

To get a data attribute through the dataset object, get the property by the part of the attribute name after data- (note that dashes are converted to camelCase).



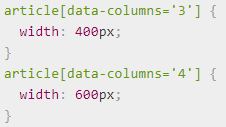
Each property is a string and can be read and written. In the above case setting article.dataset.columns = 5 would change that attribute to "5".

**CSS access**

As data attributes are plain HTML attributes, you can even access them from CSS. For example to show the parent data on the article you can use generated content in CSS with the attr() function:



You can also use the attribute selectors in CSS to change styles according to the data:



Data attributes can also be stored to contain information that is constantly changing. Using the CSS selectors and JavaScript access here this allows to build some nifty effects without having to write own display routines.

Data values are strings. Number values must be quoted in the selector for the styling to take effect.

1. [How to associate human readable content with arbitrary computer data structures](https://developer.mozilla.org/en-US/docs/Learn/HTML/Howto/Associate_human_readable_content_with_arbitrary_computer_data_structures)

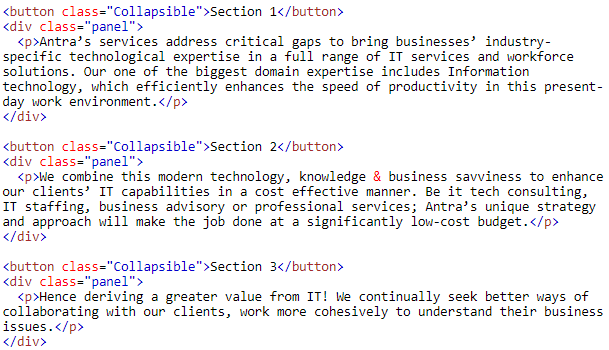
### Interactivity

1. [How to create collapsible content with HTML](https://developer.mozilla.org/en-US/docs/Learn/HTML/Howto/Create_collapsible_content_with_HTML)

Create collapsible content with HTML.

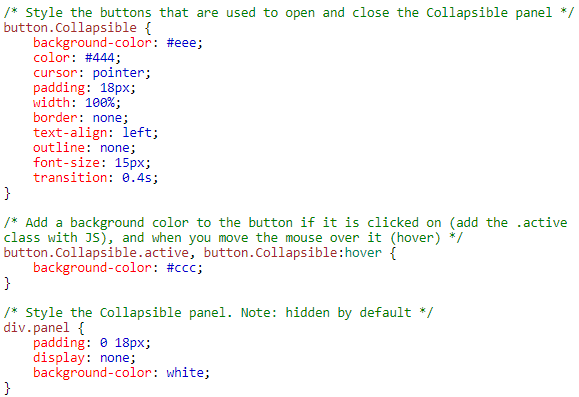
**Step 1: Add HTML**

**Example**

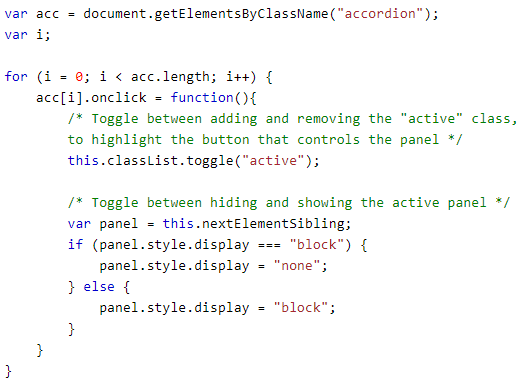
****

**Step 2: Add CSS**

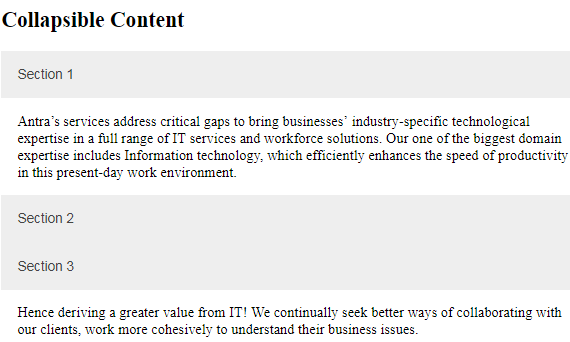
**Style the Collapsible:**



**Step 3: Add JavaScript**

****

**Output:**

****

1. [How to add context menus to a webpage](https://developer.mozilla.org/en-US/docs/Learn/HTML/Howto/Add_context_menus_to_a_webpage)

A context menu is a menu that appears upon user interaction, such as a right-click. HTML5 now allows us to customize this menu.

**Example**

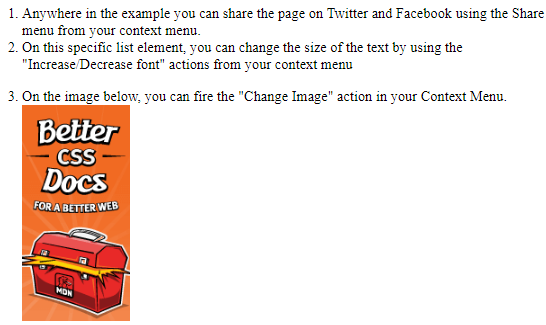
**HTML**

****

**JavaScript**

****

**Output**

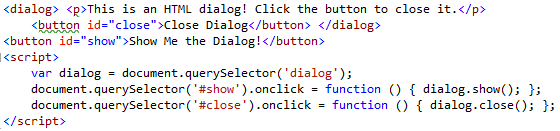
****

1. [How to create dialog boxes with HTML](https://developer.mozilla.org/en-US/docs/Learn/HTML/Howto/Create_dialog_boxes_with_HTML)

The <dialog> tag defines a dialog box or window.

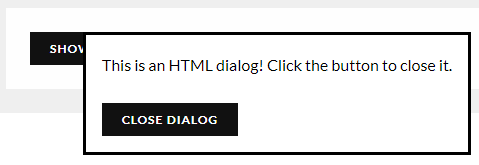
The <dialog> element makes it easy to create popup dialogs and modals on web page.

**Example**

****

**Output**

****

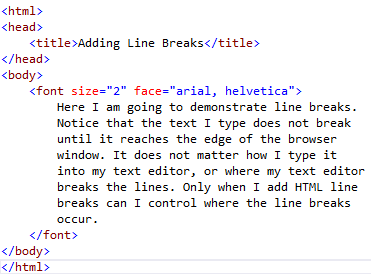
****

### Advanced text semantics

1. [How to take control of HTML line breaking](https://developer.mozilla.org/en-US/docs/Learn/HTML/Howto/Take_control_of_HTML_line_breaking)

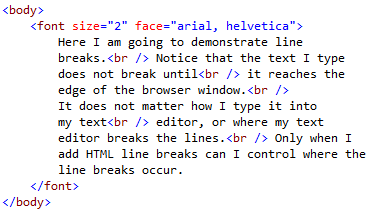
**Add a Few Lines of Text**

To create a line break on Web page, first type several sentences in text editor.



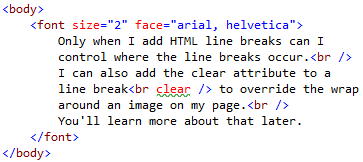
**Add the Line Break Tag**

To create a line break, type <br /> (br, a space, and a forward slash). The space and forward slash are the special closing portion of the line break tag. Now type a few more sentences.



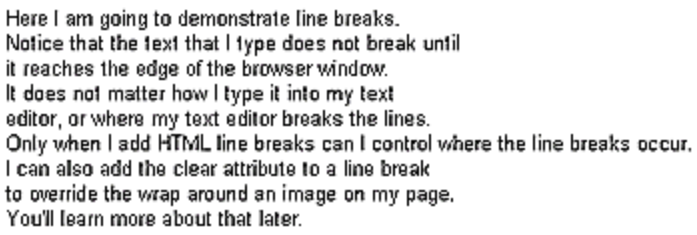
**Add the clear Attribute**

You can use the clear attribute of the line break tag to override the specified text wrap. To use the clear attribute, type <br clear />. Notice that the clear attribute is different from other attributes we've used in that it does not require a value.



**Test the Page**

Now that you've added line breaks, save page and open it in test browser. Notice that the line breaks cause the browser to move to the next line before showing the text that comes next. Also notice that the line break does not cause a paragraph break. It merely moves the text down to the next line without inserting additional space.



1. How to mark changes (added and removed text)

**HTML <ins> Tag**

The <ins> tag defines a text that has been inserted into a document.

**HTML <del> Tag**

The <del> tag defines text that has been deleted from a document.

**Example**

A text with a deleted part, and a new, inserted part:



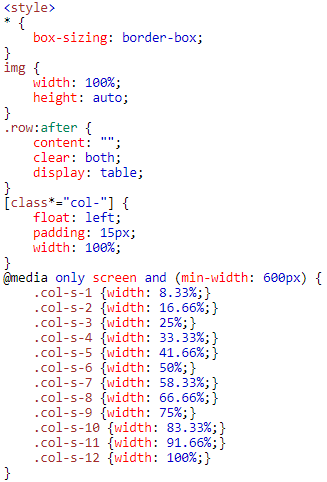
**Output:**

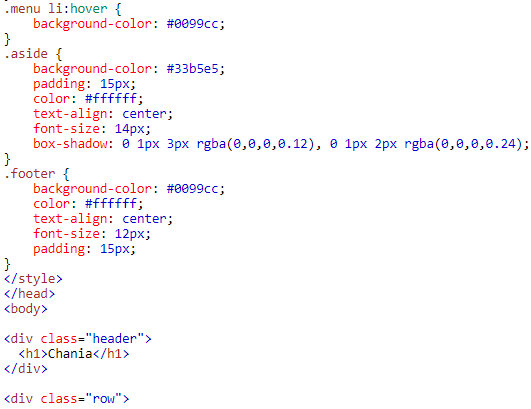
****

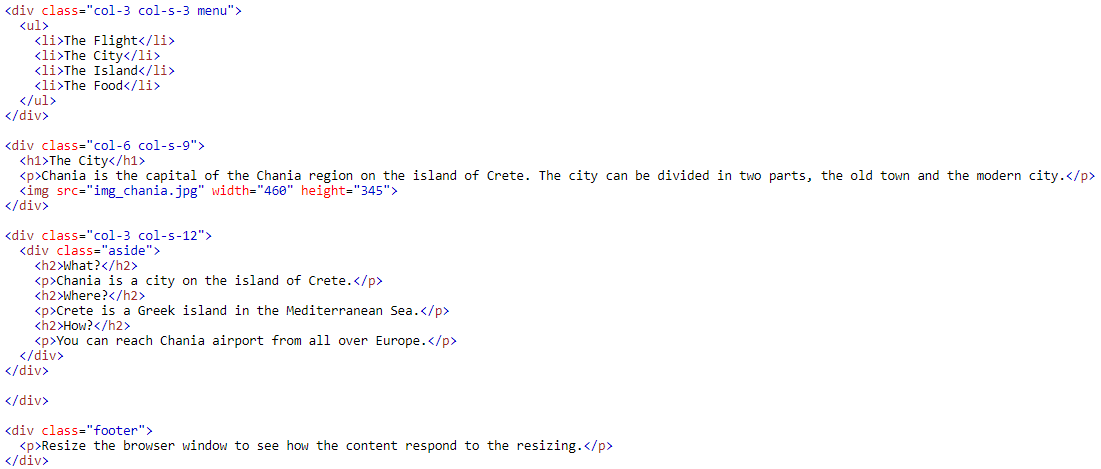
### Advanced images & multimedia

1. [How to add responsive image to a webpage](https://developer.mozilla.org/en-US/docs/Learn/HTML/Multimedia_and_embedding/Responsive_images)

**Add an Image to The Example Web Page**

****

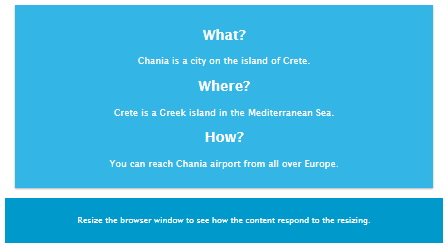
****

****

**Output**

****

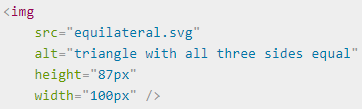




1. [How to add vector image to a webpage](https://developer.mozilla.org/en-US/docs/Learn/HTML/Multimedia_and_embedding/Adding_vector_graphics_to_the_Web)

**The quick way:<img>**

To embed an SVG via an <img> element, you just need to reference it in the src attribute. You will need a height or a width attribute (or both if your SVG has no inherent aspect ratio).



**Pros**

* Quick, familiar image syntax with built-in text equivalent available in the alt attribute.
* You can make the image into a hyperlink easily by nesting the <img> inside an <a> element.

**Cons**

* You cannot manipulate the image with JavaScript.
* If you want to control the SVG content with CSS, you must include inline CSS styles in your SVG code. (External stylesheets invoked from the SVG file take no effect.)
* You cannot restyle the image with CSS pseudoclasses (like :focus).

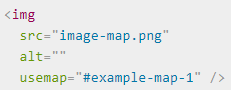
1. [How to add a hit map on top of an image](https://developer.mozilla.org/en-US/docs/Learn/HTML/Howto/Add_a_hit_map_on_top_of_an_image)

**Step 1: The image**

* The image must make it clear what happens when people follow image links. alt text is mandatory, of course, but many people never see it.
* The image must clearly indicate where hotspots begin and end.
* Hotspots must be large enough to tap comfortably, at any viewport size. How large is large enough? 72 × 72 CSS pixels is a good minimum, with additional generous gaps between touch targets. The map of the world at 50languages.com (as of time of writing) illustrates the problem perfectly. It’s much easier to tap Russia or North America than Albania or Estonia.

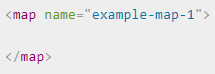
You insert image much the same way as always (with an <img> element and alt text). If the image is only present as a navigation device, you may write alt="", provided furnish appropriate alt text in the <area> elements later on.

You will need a special usemap attribute. Come up with a unique name, containing no spaces, for your image map. Then assign that name (preceded by a hash) as the value for the usemap attribute:



**Step 2: Activate your hotspots**

In this step, put all your code inside a **<map>** element. **<map>** only needs one attribute, the same map **name** as you used in your **usemap** attribute above:



Inside the **<map>** element, we need **<area>** elements. An **<area>** element corresponds to a single hotspot. To keep keyboard navigation intuitive, make sure the source order of **<area>** elements corresponds to the visual order of hotspots.

**<area>** elements are empty elements, but do require four attributes:

**shape**

**coords**

**shape** takes one of four values: **circle, rect, poly**, and **default**. (A **default <area>** occupies the entire image, minus any other hotspots you’ve defined.) The shape you choose determines the coordinate information you’ll need to provide in coords.

* For a circle, provide the center's x and y coordinates, followed by the length of the radius.
* For a rectangle, provide the x/y coordinates of the upper-left and bottom-right corners.
* For a polygon, to provide the x/y coordinates of each corner (so, at least six values).

Coordinates are given in CSS pixels.

In case of overlap, source order carries the day.

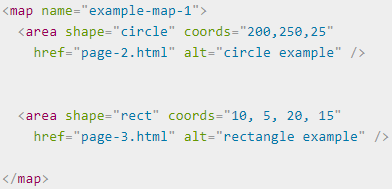
**href**

The URL of the resource you’re linking to. You may leave this attribute blank if you don’t want the current area to link anywhere (say, if you’re making a hollow circle.)

**alt**

A mandatory attribute, telling people where the link goes or what it does. alt text only displays when the image is unavailable. Please refer to our guidelines for writing accessible link text.

You may write alt="" if the href attribute is blank and the entire image already has an alt attribute.



**Step 3: Make sure it works for everybody**

You aren’t done until you test image maps rigorously on many browsers and devices. Try following links with your keyboard alone. Try turning images off.

If your image map is wider than about 240px, you’ll need to make further adjustments to make your website responsive. It's not enough to resize the image for small screens, because the coordinates stay the same and no longer match the image.

### Internationalization

1. [How to add multiple languages into a single webpage](https://developer.mozilla.org/en-US/docs/Learn/HTML/Howto/Add_multiple_languages_into_a_single_webpage)
2. [How to handle Japanese ruby characters](https://developer.mozilla.org/en-US/docs/Learn/HTML/Howto/Handle_Japanese_ruby_characters)

If you see something like ÃƒÂ¤ÃƒÂ¶ÃƒÂ¼ÃƒÅ on your HTML page, your text source is probably correct utf-8, but your browser is not set to displaying it as such. Use this metatag as the first metatag in the head section of html pages to define the utf-8 characterset:   
<meta http-equiv="Content-Type" content="text/html; charset=utf-8>  
Or use the shorter tag for new (HTML5 compliant) browsers:  
<meta charset="utf-8">  
They should have the same effect: telling the browser which character encoding to use. UTF-8 contains just about every alphabet used in the world.

In HTML5 there is a provision for the use of small "Ruby characters" (furigana) sometimes used for pronunciation guidance of Kanji characters:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **hiragana** | | **katakana** | | **romaji** | |
| とう | きょう | トウ | キョウ | tō | kyō |
| 東 | 京 | 東 | 京 | 東 | 京 |

("Tokio")

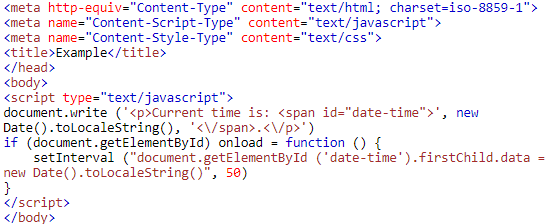
**HTML5 markup:** <ruby>東<rp>(</rp><rt>とう</rt><rp>)</rp>京<rp>(</rp><rt>きょう</rt><rp>)</rp></ruby>  
**Rendering:**

If the browser (possibly helped by a respective plugin) supports the Ruby tags the Ruby characters are shown as small characters above the Kanji, otherwise they appear in parenthesis directly after the character.

1. [How to display time and date with HTML](https://developer.mozilla.org/en-US/docs/Learn/HTML/Howto/Display_time_and_date_with_HTML)

**Step 1:** Open a simple text editor such as Notepad or Notepad++ on Windows or TextEdit on a Mancintosh.

**Step 2:** Begin an HTML document consisting of the following code.

****

**Step 3:** Save the page. Personalize the file's name to make it easy to distinguish when you have you viewer read the page. This will give you a HTML link instead of a JavaScript.

**Step 4:** Open the file. You can see digital clock in HTML.



### Performance

1. [How to authoring fast-loading HTML pages](https://developer.mozilla.org/en-US/docs/Learn/HTML/Howto/Author_fast-loading_HTML_pages)

Tips for authoring fast-loading HTML pages

**Reduce page weight**

Reducing page weight through the elimination of unnecessary whitespace and comments, commonly known as minimization, and by moving inline script and CSS into external files, can improve download performance with minimal need for other changes in the page structure.

Tools such as HTML Tidy can automatically strip leading whitespace and extra blank lines from valid HTML source. Other tools can "compress" JavaScript by reformatting the source or by obfuscating the source and replacing long identifiers with shorter versions.

**Minimize the number of files**

Reducing the number of files referenced in a web page lowers the number of HTTP connections required to download a page, thereby reducing the time for these requests to be sent, and for their responses to be received.

Depending on a browser's cache settings, it may send a request with the If-Modified-Since header for each referenced file, asking whether the file has been modified since the last time it was downloaded. Too much time spent querying the last modified time of the referenced files can delay the initial display of the web page, since the browser must check the modification time for each of these files, before rendering the page.

**Reduce domain lookups**

Since each separate domain costs time in a DNS lookup, the page load time will grow along with the number of separate domains appearing in CSS link(s) and JavaScript and image src(es).

This may not always be practical; however, you should always take care to use only the minimum necessary number of different domains in pages.

**Cache reused content**

Make sure that any content that can be cached, is cached, and with appropriate expiration times.

In particular, pay attention to the Last-Modified header. It allows for efficient page caching; by means of this header, information is conveyed to the user agent about the file it wants to load, such as when it was last modified. Most web servers automatically append the Last-Modified header to static pages (e.g. .html, .css), based on the last-modified date stored in the file system. With dynamic pages (e.g. .php, .aspx), this, of course, can't be done, and the header is not sent.

So, in particular for pages which are generated dynamically, a little research on this subject is beneficial. It can be somewhat involved, but it will save a lot in page requests on pages which would normally not be cacheable.

**Optimally order the components of the page**

Download page content first, along with any CSS or JavaScript that may be required for its initial display, so that the user gets the quickest apparent response during the page loading. This content is typically text, and can therefore benefit from text compression in transit, thus providing an even quicker response to the user.

Any dynamic features that require the page to complete loading before being used, should be initially disabled, and then only enabled after the page has loaded. This will cause the JavaScript to be loaded after the page contents, which will improve the overall appearance of the page load.

**Reduce the number of inline scripts**

Inline scripts can be expensive for page loading, since the parser must assume that an inline script could modify the page structure while parsing is in progress. Reducing the use of inline scripts in general, and reducing the use of document.write() to output content in particular, can improve overall page loading. Use modern AJAX methods to manipulate page content for modern browsers, rather than the older approaches based on document.write().

**Use modern CSS and valid markup**

Use of modern CSS reduces the amount of markup, can reduce the need for (spacer) images, in terms of layout, and can very often replace images of stylized text -- that "cost" much more than the equivalent text-and-CSS.

Using valid markup has other advantages. First, browsers will have no need to perform error-correction when parsing the HTML (this is aside from the philosophical issue of whether to allow format variation in user input, and then programmatically "correct" or normalize it; or whether, instead, to enforce a strict, no-tolerance input format).

Moreover, valid markup allows for the free use of other tools which can pre-process your web pages. For example, HTML Tidy can remove whitespace and optional ending tags; however, it will refuse to run on a page with serious markup errors.

**Minify and compress SVG assets**

SVG produced by most drawing applications often contains unnecessary metadata which can be removed. Configure your servers apply gzip compression for SVG assets.

**Choose your user-agent requirements wisely**

To achieve the greatest improvements in page design, make sure that reasonable user-agent requirements are specified for projects. Do not require your content to appear pixel-perfect in all browsers, especially not in down-version browsers.

Ideally, your basic minimum requirements should be based on the consideration of modern browsers that support the relevant standards. This can include recent versions of Firefox, Internet Explorer, Google Chrome, Opera, and Safari.

**Use async and defer, if possible**

Make the JavaScript scripts such that they are compatible with both the async and the defer and use async whenever possible, especially if you have multiple script tags.

With that, the page can stop rendering while JavaScript is still loading. Otherwise, the browser will not render anything that is after the script tags that do not have these attributes.