# Coding Guidelines | HTML5 CSS Coding Guidelines

**Omit the protocol from embedded resources.**

Omit the protocol portion (http:, https:) from URLs pointing to images and other media files, style sheets, and scripts unless the respective files are not available over both protocols.

Omitting the protocol—which makes the URL relative—prevents mixed content issues and results in minor file size savings.

<!-- Not recommended -->

<script src="[https://www.google.com/js/gweb/analytics/autotrack.js](http://www.google.com/js/gweb/analytics/autotrack.js)"></script>

<!-- Recommended -->

<script src="//[www.google.com/js/gweb/analytics/autotrack.js](http://www.google.com/js/gweb/analytics/autotrack.js)"></script>

/\* Not recommended \*/

.example {

background: url([https://www.google.com/images/example](http://www.google.com/images/example)%3B));

}

/\* Recommended \*/

.example {

background: url(//[www.google.com/images/example);](http://www.google.com/images/example)%3B)

}

**Use UTF-8 :-**

Make sure your editor uses UTF-8 as character encoding, without a byte order mark.

Specify the encoding in HTML templates and documents via <meta charset="utf-8">. Do not specify the encoding of style sheets as these assume UTF-8.

**Explain code as needed, where possible.**

Use comments to explain code: What does it cover, what purpose does it serve, why is respective solution used or preferred?

**Semantics**

Use HTML according to its purpose.

Use elements (sometimes incorrectly called “tags”) for what they have been created for. For example, use heading elements for headings, p elements for paragraphs, a elements for anchors, etc.

Using HTML according to its purpose is important for accessibility, reuse, and code efficiency reasons.

<!-- Not recommended -->

<div onclick="goToRecommendations();">All recommendations</div>

<!-- Recommended -->

<a href="recommendations/">All recommendations</a>

Separation of Concerns

Separate structure from presentation from behaviour.

Strictly keep structure (markup), presentation (styling), and behavior (scripting) apart, and try to keep the interaction between the three to an absolute minimum.

That is, make sure documents and templates contain only HTML and HTML that is solely serving structural purposes. Move everything presentational into style sheets, and everything behavioral into scripts.

In addition, keep the contact area as small as possible by linking as few style sheets and scripts as possible from documents and templates.

Separating structure from presentation from behavior is important for maintenance reasons. It is always more expensive to change HTML documents and templates than it is to update style sheets and scripts.

<!-- Not recommended -->

<!DOCTYPE html>

<title>HTML sucks</title>

<link rel="stylesheet" href="base.css" media="screen">

<link rel="stylesheet" href="grid.css" media="screen">

<link rel="stylesheet" href="print.css" media="print">

<h1 style="font-size: 1em;">HTML sucks</h1>

<p>I’ve read about this on a few sites but now I’m sure:

<u>HTML is stupid!!1</u>

<center>I can’t believe there’s no way to control the styling of my website without doing everything all over again!</center>

<!-- Recommended -->

<!DOCTYPE html>

<title>My first CSS-only redesign</title>

<link rel="stylesheet" href="default.css">

<h1>My first CSS-only redesign</h1>

<p>I’ve read about this on a few sites but today I’m actually

doing it: separating concerns and avoiding anything in the HTML of my website that is presentational.

<p>It’s awesome!

**General Formatting**

Use a new line for every block, list, or table element, and indent every such child element.

Independent of the styling of an element (as CSS allows elements to assume a different role per display property), put every block, list, or table element on a new line.

Also, indent them if they are child elements of a block, list, or table element.

(If you run into issues around white space between list items it’s acceptable to put all li elements in one line. A linter is encouraged to throw a warning instead of an error.)

<blockquote>

<p><em>Space</em>, the final frontier.</p>

</blockquote>

<ul>

<li>Moe

<li>Larry

<li>Curly

</ul>

<table>

<thead>

<tr>

<th scope="col">Income

<th scope="col">Taxes

<tbody>

<tr>

<td>$ 5.00

<td>$ 4.50

</table>

**Optional Tags**

Omit optional tags (optional).

For file size optimization and scannability purposes, consider omitting optional tags. The HTML5 specification defines what tags can be omitted.

(This approach may require a grace period to be established as a wider guideline as it’s significantly different from what web developers are typically taught. For consistency and simplicity reasons it’s best served omitting all optional tags, not just a selection.)

<!-- Not recommended -->

<!DOCTYPE html>

<html>

<head>

<title>Spending money, spending bytes</title>

</head>

<body>

<p>Sic.</p>

</body>

</html>

<!-- Recommended -->

<!DOCTYPE html>

<title>Saving money, saving bytes</title>

<p>Qed.

**Closing Tags :-**

Include closing tags for all container-type elements. For example:

<p>

Park University, founded in 1625 by the Franciscan Monks, quickly gained national attention for its efforts in the fields of Alchemy and Computer Science.

</p>

This is not required by the HTML5 standard, but in the interest of consistency and readability, you should do it.

**Use lowercase characters for all tags and attributes :-**

This is not required by the HTML5 standard, but in the interest of consistency and readability, you should do it.

**Enclose all attribute values in double quotes :-**

<div style="color:#0000FF">

<h3>This is a heading</h3>

<p>This is a paragraph.</p>

</div>

**Naming Convention of name attribute :-**

Because the name attribute can be used to identify elements in JavaScript, and JavaScript coding conventions suggests using camel case for multi-word variable names, use camel case for values for the name attribute.

**Blank lines:**

Insert blank lines to separate large logical chunks of code. Insert a blank line between the head and body sections. Do not insert a blank line unless there's a reason to do so.

**Indentations:**

* Indentations should use two spaces. Do not use tabs.



Avoid long lines:

Lines should be no longer than 80 characters.



Top of the document

Include the following line at the top of your HTML5 documents:

**<!DOCTYPE html>**

This will tell the browser to use the HTML5 version of HTML (because other versions of HTML use slightly different forms of the doctype tag).

**Use Lower Case File Names**

Some web servers (Apache, Unix) are case sensitive about file names: "london.jpg" cannot be accessed as "London.jpg".

Other web servers (Microsoft, IIS) are not case sensitive: "london.jpg" can be accessed as "London.jpg" or "london.jpg".

If you use a mix of upper and lower case, you have to be extremely consistent.

If you move from a case insensitive to a case sensitive server, even small errors will break your web!

To avoid these problems, always use lower case file names.

**Forms**

Form fields must always include a <label> element with a "for" attribute matching the "id" on the input. This helps accessibility by focusing the input when the label is clicked, it also helps screen readers match labels to their respective inputs.

<label for="field-email">email</label>

<input type="email" id="field-email" name="email" value="" />

Each <input> should have an "id" that is unique to the page. It does not have to match the "name" attribute.

**Descriptive Meta Tags:**

Meta elements are typically used to specify page description, keywords, author of the document, last modified, and other metadata and makes your web page more meaningful for user agents like search engine spiders.

Hide Copy Code

<head>

...

<meta name="keywords" content="HTML, CSS, XML, XHTML, JavaScript">

</head>

**Avoid Inline Styles**

HTML is not HyperText Styling language, CSS is there for a reason which leaves the inline styles as a resource when required.

Hide Copy Code

<h1 style="color:blue;">Is is really necessary?</h1>

**Have CSS Links in the HEAD Tag**

It has been tested and it seems that pages render faster when moving stylesheet tags to the document head:

**Placing JavaScript Files**

If you have JavaScript files that are exclusively to add functionality to a page, it's better to place it at the end of the HTML file, right before the closing body tag. This will allows the page to load faster.

**Use web storage in place of cookies**

While cookies have been used to track unique user data for years, they have serious disadvantages. The largest flaw is that all of your cookie data is added to every HTTP request header. This can end up having a measurable impact on response time, especially during XHRs. So a best practice is to reduce cookie size. In HTML5 we can do better than that: use sessionStorage and localStorage in place of cookies.

**Never Use Inline Javascript.**

Another common practice years ago was to place JS commands directly within tags. This was very common with simple image galleries. Essentially, a "onclick" attribute was appended to the tag. The value would then be equal to some JS procedure. Needless to say, you should never, ever do this. Instead, transfer this code to an external JS file and use "addEventListener/attachEvent" to "listen" for your desired event. Or, if using a framework like jQuery, just use the "click" method.

**Head Container**

The following items should be included in the order shown.

**To avoid content being interpreted incorrectly, include the following charset meta element:**

<meta charset="utf-8">



For grading purposes, I need to know your name, so you are required to include an author meta element. For example:

<meta name="author" content="John Dean">



As an option, to help search engines find your pages, you may also want to include description and keywords meta elements. For example:

<meta name="description"

content="This web page presents Dean family highlights.">

<meta name="keywords" content="Dean family">



Include a meaningful title element on each page. For example:

<title>Dean Family</title>

Rationale: Helps search engine functionality.



For external style sheets, use the link element. For example:

<link rel="stylesheet" href="style.css">



To ensure proper interpretation, and correct search engine indexing, both the language and the character encoding should be defined as early as possible in a document:

<!DOCTYPE html>

<html lang="en-US">

<head>

<meta charset="UTF-8">

<title>HTML5 Syntax and Coding Style</title>

</head>

Body Container



Images and colors:

Include width and height attributes for all img elements.

Rationale: Faster page loading.

Include the alt attribute for all img elements. Use alt="" for decorative images.

Rationale: Alternate text serves as content when an image cannot be rendered or viewed normally.

Do not use animated gifs or use them sparingly.

Rationale: Animations are annoying.

Omitting <html> and <body>?

In the HTML5 standard, the <html> tag and the <body> tag can be omitted.

The <html> element is the document root. It is the recommended place for specifying the page language:

**<!DOCTYPE html>**

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

Declaring a language is important for accessibility applications (screen readers) and search engines.

Omitting <html> or <body> can crash DOM and XML software.

Omitting <body> can produce errors in older browsers (IE9).



Omitting <head>?

In the HTML5 standard, the <head> tag can also be omitted.

By default, browsers will add all elements before <body>, to a default <head> element.

You can reduce the complexity of HTML, by omitting the <head> tag:

Example

<!DOCTYPE html>

<html>

<title>Page Title</title>

<body>

<h1>This is a heading</h1>

<p>This is a paragraph.</p>

</body>

</html>

We do not recommend omitting the <head> tag.

Omitting tags is unfamiliar to web developers. It needs time to be established as a guideline.

STYLE SHEETS:

ID and Class Naming

Use meaningful or generic ID and class names.

Instead of presentational or cryptic names, always use ID and class names that reflect the purpose of the element in question, or that are otherwise generic.

Names that are specific and reflect the purpose of the element should be preferred as these are most understandable and the least likely to change.

Generic names are simply a fallback for elements that have no particular or no meaning different from their siblings. They are typically needed as “helpers.”

Using functional or generic names reduces the probability of unnecessary document or template changes.

/\* Not recommended: meaningless \*/ #yee-1901 {}

/\* Not recommended: presentational \*/

.button-green {}

.clear {}

/\* Recommended: specific \*/ #gallery {}

#login {}

.video {}

/\* Recommended: generic \*/

.aux {}

.alt {}

Type Selectors

Avoid qualifying ID and class names with type selectors.

Unless necessary (for example with helper classes), do not use element names in conjunction with IDs or classes.

Avoiding unnecessary ancestor selectors is useful for performance reasons.

/\* Not recommended \*/ ul#example {} div.error {}

/\* Recommended \*/ #example {}

.error {}

Hexadecimal Notation

Use 3 character hexadecimal notation where possible.

For color values that permit it, 3 character hexadecimal notation is shorter and more succinct.

/\* Not recommended \*/ color: #eebbcc;

/\* Recommended \*/ color: #ebc;

Block Content Indentation

Indent all block content.

Indent all block content, that is rules within rules as well as declarations, so to reflect hierarchy and improve understanding.

@media screen, projection {

html {

background: #fff;

color: #444;

}

}

Declaration Stops

Use a semicolon after every declaration.

End every declaration with a semicolon for consistency and extensibility reasons.

/\* Not recommended \*/

.test {

display: block; height: 100px

}

/\* Recommended \*/

.test {

display: block; height: 100px;

}

Property Name Stops

Use a space after a property name’s colon.

Always use a single space between property and value (but no space between property and colon) for consistency reasons.

/\* Not recommended \*/ h3 {

font-weight:bold;

}

/\* Recommended \*/ h3 {

font-weight: bold;

}

Declaration Block Separation

Use a space between the last selector and the declaration block.

Always use a single space between the last selector and the opening brace that begins the declaration block.

The opening brace should be on the same line as the last selector in a given rule.

/\* Not recommended: missing space \*/ #video{

margin-top: 1em;

}

/\* Not recommended: unnecessary line break \*/ #video

{

margin-top: 1em;

}

/\* Recommended \*/ #video {

margin-top: 1em;

}

Rule Separation

Separate rules by new lines.

Always put a blank line (two line breaks) between rules.

html {

background: #fff;

}

body {

margin: auto; width: 50%;

}

In an embedded style element and also in a link element to an external style sheet, include a type attribute with a value of "text/css". Here are two separate examples:

<style>

...

</style>

<link rel="stylesheet" href="style.css">

CSS is the default type for a style element, so, functionally, there’s no difference between including and omitting type="text/css". To avoid clutter, omit it.

Comments:

You should supply comments within your CSS code for any rules that are non-intuitive. If the comment is short, insert it to the right of the rule that the comment applies to.

If the comment is long, insert it immediately above the rule(s) that the comment applies to. To give the comment more prominence, insert a blank line above the comment.

Short rules:

* If you have just one property-value pair, or you have just a few property-value pairs, and none of the values are font-family lists, then you may put them on one line, with a space following each interior semicolon, like this:

.opening-statement {font-style: italic; color: blue;}

.closing-statement {font-family: "Times New Roman", Times, serif;}

With one-line rules, insert a blank space before the opening brace. There should be no blank spaces to the right of the opening brace and to the left of the closing brace.



Long rules:

* If you have a rule with a long heading and/or lots of property-value pairs, use block-style notation. For example:

body {

background-color: white; color: black;

a:visited: green; a:link: red;

font-family: "Arial Black", Helvetica, sans-serif;

}

* With block-style notation, place the opening brace on the same line as the selector, separated from the selector by one space.

Insert a blank space after a CSS property's colon. There should be no blank space before the colon.

For a font family list, insert a space after each comma.

Use hyphens to separate multi-word class values and multi-word id values.

Include a semicolon after the last property-value pair.

* Although CSS standards allow you to omit the semicolon after the last property-value pair, you should include it. That way, if another rule is added later on, there will be less likelihood of accidentally forgetting the semicolon to separate the prior property-value pair from the new one.