Google HTML/CSS Style Guide

Revision 2.23

Each style point has a summary for which additional information is available by toggling the accompanying arrow button that looks this way: ▽ . You may toggle all summaries with the big arrow button:

▽ Toggle all summaries

Table of Contents

Important Note

Displaying Hidden Details in This Guide

link ▽ This style guide contains many details that are initially hidden from view. They are marked by the triangle icon, which you see here on your left. Click it now. You should see “Hooray” appear below.

Hooray! Now you know you can expand points to get more details. Alternatively, there’s a “toggle all” at the top of this document.

Background

This document defines formatting and style rules for HTML and CSS. It aims at improving collaboration, code quality, and enabling supporting infrastructure. It applies to raw, working files that use HTML and CSS, including GSS files. Tools are free to obfuscate, minify, and compile as long as the general code quality is maintained.

General Style Rules

Protocol

link ▽ 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="http://www.google.com/js/gweb/analytics/autotrack.js"></script>
```

```
<!-- Recommended -->
<script src="//www.google.com/js/gweb/analytics/autotrack.js"></script>
```

```
/* Not recommended */
.example {
  background: url(http://www.google.com/images/example);
}
```

```
/* Recommended */
.example {
  background: url(//www.google.com/images/example);
}
```

General Formatting Rules

Indentation

link ▽ Indent by 2 spaces at a time.

Don’t use tabs or mix tabs and spaces for indentation.

```
<ul>
  <li>Fantastic
  <li>Great
</ul>
```

```
.example {
  color: blue;
}
```

Capitalization

link ▽ Use only lowercase.

All code has to be lowercase: This applies to HTML element names, attributes, attribute values (unless `text/CDATA`), CSS selectors, properties, and property values (with the exception of strings).

```
<!-- Not recommended -->
<A HREF="/">Home</A>
```

```
<!-- Recommended -->
<img src="google.png" alt="Google">
```

```
/* Not recommended */
color: #E5E5E5;
```

```
/* Recommended */
color: #e5e5e5;
```

Trailing Whitespace

link ▽ Remove trailing white spaces.

Trailing white spaces are unnecessary and can complicate diffs.

```
<!-- Not recommended -->
<p>What?_
```

```
<!-- Recommended -->
<p>Yes please.
```

General Meta Rules

Encoding

link ▽ Use UTF-8 (no BOM).

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.

(More on encodings and when and how to specify them can be found in Handling character encodings in HTML and CSS.)

Comments

link ▽ 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?

(This item is optional as it is not deemed a realistic expectation to always demand fully documented code. Mileage may vary heavily for HTML and CSS code and depends on the project's complexity.)

Action Items

link ▽ Mark todos and action items with `TODO`.

Highlight todos by using the keyword `TODO` only, not other common formats like `@@`.

Append a contact (username or mailing list) in parentheses as with the format `TODO(contact)`.

Append action items after a colon as in `TODO: action item`.

```
{# TODO(john.doe): revisit centering #}
<center>Test</center>
```

```
<!-- TODO: remove optional tags -->
<ul>
  <li>Apples</li>
  <li>Oranges</li>
</ul>
```

HTML Style Rules

Document Type

link ▽ Use HTML5.

HTML5 (HTML syntax) is preferred for all HTML documents: `<!DOCTYPE html>`.

(It's recommended to use HTML, as `text/html`. Do not use XHTML. XHTML, as

application/xhtml+xml, lacks both browser and infrastructure support and offers less room for optimization than HTML.)

Although fine with HTML, do not close void elements, i.e. write `<br>`, not `<br />`.

HTML Validity

link ▽ Use valid HTML where possible.

Use valid HTML code unless that is not possible due to otherwise unattainable performance goals regarding file size.

Use tools such as the W3C HTML validator to test.

Using valid HTML is a measurable baseline quality attribute that contributes to learning about technical requirements and constraints, and that ensures proper HTML usage.

```
<!-- Not recommended -->
<title>Test</title>
<article>This is only a test.
```

```
<!-- Recommended -->
<!DOCTYPE html>
<meta charset="utf-8">
<title>Test</title>
<article>This is only a test.</article>
```

Semantics

link ▽ 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>
```

Multimedia Fallback

link ▽ Provide alternative contents for multimedia.

For multimedia, such as images, videos, animated objects via `canvas`, make sure to offer alternative access. For images that means use of meaningful alternative text (`alt`) and for video and audio transcripts and captions, if available.

Providing alternative contents is important for accessibility reasons: A blind user has few cues to tell what an image is about without `@alt`, and other users may have no way of understanding what video or audio contents are about either.

(For images whose `alt` attributes would introduce redundancy, and for images whose purpose is purely decorative which you cannot immediately use CSS for, use no alternative text, as in `alt=""`.)

```
<!-- Not recommended -->
<img src="spreadsheet.png">
```

```
<!-- Recommended -->
<img src="spreadsheet.png" alt="Spreadsheet screenshot.">
```

Separation of Concerns

link ▽ Separate structure from presentation from behavior.

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!
```

Entity References

link ▽ Do not use entity references.

There is no need to use entity references like —, ”, or ☺, assuming the same encoding (UTF-8) is used for files and editors as well as among teams.

The only exceptions apply to characters with special meaning in HTML (like < and &) as well as control or "invisible" characters (like no-break spaces).

```
<!-- Not recommended -->
The currency symbol for the Euro is &ldquo;&eur;&rdquo;.
```

```
<!-- Recommended -->
The currency symbol for the Euro is "€".
```

Optional Tags

link ▽ 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.
```

type Attributes

link ▽ Omit `type` attributes for style sheets and scripts.

Do not use `type` attributes for style sheets (unless not using CSS) and scripts (unless not using JavaScript).

Specifying `type` attributes in these contexts is not necessary as HTML5 implies `text/css` and `text/javascript` as defaults. This can be safely done even for older browsers.

```
<!-- Not recommended -->
<link rel="stylesheet" href="//www.google.com/css/maia.css"
  type="text/css">
```

```
<!-- Recommended -->
<link rel="stylesheet" href="//www.google.com/css/maia.css">
```

```
<!-- Not recommended -->
<script src="//www.google.com/js/gweb/analytics/autotrack.js"
  type="text/javascript"></script>
```

```
<!-- Recommended -->
<script src="//www.google.com/js/gweb/analytics/autotrack.js"></script>
```

HTML Formatting Rules

General Formatting

link ▽ 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 whitespace 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>
```

HTML Quotation Marks

link ▽ When quoting attributes values, use double quotation marks.

Use double ("") rather than single quotation marks ('') around attribute values.

```
<!-- Not recommended -->
<a class='maia-button maia-button-secondary'>Sign in</a>
```

```
<!-- Recommended -->
<a class="maia-button maia-button-secondary">Sign in</a>
```

CSS Style Rules

CSS Validity

link ▽ Use valid CSS where possible.

Unless dealing with CSS validator bugs or requiring proprietary syntax, use valid CSS code.

Use tools such as the W3C CSS validator to test.

Using valid CSS is a measurable baseline quality attribute that allows to spot CSS code that may not have any effect and can be removed, and that ensures proper CSS usage.

ID and Class Naming

link ▽ 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 {}
```

ID and Class Name Style

link ▽ Use ID and class names that are as short as possible but as long as necessary.

Try to convey what an ID or class is about while being as brief as possible.

Using ID and class names this way contributes to acceptable levels of understandability and code efficiency.

```
/* Not recommended */
#navigation {}
.atr {}
```

```
/* Recommended */
#nav {}
.author {}
```

Type Selectors

link ▽ 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 {}
```

Shorthand Properties

link ▽ Use shorthand properties where possible.

CSS offers a variety of shorthand properties (like `font`) that should be used whenever possible, even in cases where only one value is explicitly set.

Using shorthand properties is useful for code efficiency and understandability.

```
/* Not recommended */
border-top-style: none;
font-family: palatino, georgia, serif;
font-size: 100%;
line-height: 1.6;
padding-bottom: 2em;
padding-left: 1em;
padding-right: 1em;
padding-top: 0;
```

```
/* Recommended */
border-top: 0;
font: 100%/1.6 palatino, georgia, serif;
padding: 0 1em 2em;
```

0 and Units

link ▽ Omit unit specification after “0” values.

Do not use units after `0` values unless they are required.

```
margin: 0;
padding: 0;
```

Leading 0s

link ▽ Omit leading “0”s in values.

Do not use put `0`s in front of values or lengths between -1 and 1.

```
font-size: .8em;
```

Hexadecimal Notation

link ▽ 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;
```

Prefixes

link ▽ Prefix selectors with an application-specific prefix (optional).

In large projects as well as for code that gets embedded in other projects or on external sites use prefixes (as namespaces) for ID and class names. Use short, unique identifiers followed by a dash.

Using namespaces helps preventing naming conflicts and can make maintenance easier, for example in search and replace operations.

```
.adw-help {} /* AdWords */
#maia-note {} /* Maia */
```

ID and Class Name Delimiters

link ▽ Separate words in ID and class names by a hyphen.

Do not concatenate words and abbreviations in selectors by any characters (including none at all) other than hyphens, in order to improve understanding and scannability.

```
/* Not recommended: does not separate the words “demo” and “image” */
.demoimage {}

/* Not recommended: uses underscore instead of hyphen */
.error_status {}
```

```
/* Recommended */
#video-id {}
.ads-sample {}
```

Hacks

link ▽ Avoid user agent detection as well as CSS “hacks”—try a different approach first.

It’s tempting to address styling differences over user agent detection or special CSS filters, workarounds, and hacks. Both approaches should be considered last resort in order to achieve and maintain an efficient and manageable code base. Put another way, giving detection and hacks a free pass will hurt projects in the long run as projects tend to take the way of least resistance. That is, allowing and making it easy to use detection and hacks means using detection and hacks more frequently—and more frequently is too frequently.

CSS Formatting Rules

Declaration Order

link ▽ Alphabetize declarations.

Put declarations in alphabetical order in order to achieve consistent code in a way that is easy to remember and maintain.

Ignore vendor-specific prefixes for sorting purposes. However, multiple vendor-specific prefixes for a certain CSS property should be kept sorted (e.g. -moz prefix comes before -webkit).

```
background: fuchsia;
border: 1px solid;
-moz-border-radius: 4px;
-webkit-border-radius: 4px;
border-radius: 4px;
color: black;
text-align: center;
text-indent: 2em;
```

Block Content Indentation

link ▽ 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

link ▽ 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

link ▽ 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

link ▽ 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;
}
```

Selector and Declaration Separation

link ▽ Separate selectors and declarations by new lines.

Always start a new line for each selector and declaration.

```
/* Not recommended */
a:focus, a:active {
  position: relative; top: 1px;
}
```

```
/* Recommended */
h1,
h2,
h3 {
  font-weight: normal;
  line-height: 1.2;
}
```

Rule Separation

link ▽ Separate rules by new lines.

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

```
html {
  background: #fff;
}

body {
  margin: auto;
  width: 50%;
}
```

CSS Quotation Marks

link ▽ Use single quotation marks for attribute selectors and property values.

Use single (`''`) rather than double (`""`) quotation marks for attribute selectors or property values. Do not use quotation marks in URI values (`url()`).

Exception: If you do need to use the `@charset` rule, use double quotation marks—single quotation marks are not permitted.

```
/* Not recommended */
@import url("//www.google.com/css/maia.css");
```

```
html {
  font-family: "open sans", arial, sans-serif;
}
```

```
/* Recommended */
@import url(//www.google.com/css/maia.css);

html {
  font-family: 'open sans', arial, sans-serif;
}
```

CSS Meta Rules

Section Comments

link ▽ Group sections by a section comment (optional).

If possible, group style sheet sections together by using comments. Separate sections with new lines.

```
/* Header */

#adw-header {}

/* Footer */

#adw-footer {}

/* Gallery */

.adw-gallery {}
```

Parting Words

Be consistent.

If you're editing code, take a few minutes to look at the code around you and determine its style. If they use spaces around all their arithmetic operators, you should too. If their comments have little boxes of hash marks around them, make your comments have little boxes of hash marks around them too.

The point of having style guidelines is to have a common vocabulary of coding so people can concentrate on what you're saying rather than on how you're saying it. We present global style rules here so people know the vocabulary, but local style is also important. If code you add to a file looks drastically different from the existing code around it, it throws readers out of their rhythm when they go to read it. Avoid this.

Revision 2.23