Google HTML/CSS Style Guide

Table of Contents

1 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 Sass and GSS files. Tools are free to obfuscate, minify, and compile as long as the general code quality is maintained.

2 General

2.1 General Style Rules

2.1.1 Protocol

Use HTTPS for embedded resources where possible.

Always use HTTPS (`https:`) for images and other media files, style sheets, and scripts, unless the respective files are not available over HTTPS.

```
<!-- Not recommended: omits the protocol -->
<script src="//ajax.googleapis.com/ajax/libs/jquery/3.4.0/jquery.min.js"></script

<!-- Not recommended: uses HTTP -->
<script src="http://ajax.googleapis.com/ajax/libs/jquery/3.4.0/jquery.min.js"></s
```

```
<!-- Recommended -->
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.0/jquery.min.js"></s
```

```
/* Not recommended: omits the protocol */
@import '//fonts.googleapis.com/css?family=Open+Sans';

/* Not recommended: uses HTTP */
@import 'http://fonts.googleapis.com/css?family=Open+Sans';
```

```
/* Recommended */
@import 'https://fonts.googleapis.com/css?family=Open+Sans';
```

2.2 General Formatting Rules

2.2.1 Indentation

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;
}
```

2.2.2 Capitalization

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;
```

2.2.3 Trailing Whitespace

Remove trailing white spaces.

Trailing white spaces are unnecessary and can complicate diffs.

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

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

2.3 General Meta Rules

2.3.1 Encoding

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.)

2.3.2 Comments

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.)

2.3.3 Action Items

Mark todos and action items with `TODO`.

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

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

```
{# TODO: Revisit centering. #}
<center>Test</center>
```

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

3 HTML

3.1 HTML Style Rules

3.1.1 Document Type

Use `<!doctype html>`.

Always put your HTML in no-quirks mode by including `<!doctype html>` at the beginning of the document.

A document without a doctype is rendered in “quirks mode”, and one with a different doctype may be rendered in “limited-quirks mode”. These modes don’t follow the widely-understood, widely-documented behavior for various core HTML and CSS constructs, and are likely to cause subtle failures and incompatibilities especially when re-using code that expects no-quirks mode.

3.1.2 HTML Validity

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>
```

3.1.3 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>
```

3.1.4 Multimedia Fallback

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.">
```

3.1.5 Separation of Concerns

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!
```

3.1.6 Entity References

Do not use entity references.

There is no need to use entity references like `&mdash;`, `&rdquo;`, or `&#x263a;`, 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 "€".
```

3.1.7 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.
```

3.1.8 `type` Attributes

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="https://www.google.com/css/maia.css"
    type="text/css">
```

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

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

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

3.1.9 `id` Attributes

Avoid unnecessary `id` attributes.

Prefer `class` attributes for styling and `data` attributes for scripting.

Where `id` attributes are strictly required, always include a hyphen in the value to ensure it does not match the JavaScript identifier syntax, e.g. use `user-profile` rather than just `profile` or `userProfile`.

When an element has an `id` attribute, browsers will make that available as a named property on the global `window` prototype, which may cause unexpected behavior. While `id` attribute values containing a hyphen are still available as property names, these cannot be referenced as global JavaScript variables.

```
<!-- Not recommended: `window.userProfile` will resolve to reference the <div> no
<div id="userProfile"></div>
```

```
<!-- Recommended: `id` attribute is required and its value includes a hyphen -->
<div aria-describedby="user-profile">
  …
  <div id="user-profile"></div>
```

```
  …
</div>
```

3.2 HTML Formatting Rules

3.2.1 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 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>
```

3.2.2 HTML Line-Wrapping

Break long lines (optional).

While there is no column limit recommendation for HTML, you may consider wrapping long lines if it significantly improves readability.

When line-wrapping, each continuation line should be indented to distinguish wrapped attributes from child elements. Lines should be wrapped consistently within a project, ideally enforced by automated code formatting tools.

```
<button
  mat-icon-button
  color="primary"
  class="menu-button"
  (click)="openMenu()"
>
  <mat-icon>menu</mat-icon>
</button>
```

```
<button mat-icon-button color="primary" class="menu-button"
    (click)="openMenu()">
  <mat-icon>menu</mat-icon>
</button>
```

```
<button
    mat-icon-button
    color="primary"
    class="menu-button"
    (click)="openMenu()">
  <mat-icon>menu</mat-icon>
</button>
```

```
<button mat-icon-button
        color="primary"
        class="menu-button"
        (click)="openMenu()">
  <mat-icon>menu</mat-icon>
</button>
```

3.2.3 HTML Quotation Marks

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>
```

4 CSS

4.1 CSS Style Rules

4.1.1 CSS Validity

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.

4.1.2 Class Naming

Use meaningful or generic class names.

Instead of presentational or cryptic names, always use 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 {}
```

4.1.3 Class Name Style

Use class names that are as short as possible but as long as necessary.

Try to convey what a class is about while being as brief as possible.

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

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

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

4.1.4 Class Name Delimiters

Separate words in 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 {}
```

4.1.5 Prefixes

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 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 */
```

4.1.6 Type Selectors

Avoid qualifying class names with type selectors.

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

Avoiding unnecessary ancestor selectors is useful for performance reasons.

```
/* Not recommended */
ul.example {}
div.error {}
```

```
/* Recommended */
.example {}
.error {}
```

4.1.7 ID Selectors

Avoid ID selectors.

ID attributes are expected to be unique across an entire page, which is difficult to guarantee when a page contains many components worked on by many different engineers. Class selectors should be preferred in all situations.

```
/* Not recommended */
#example {}
```

```
/* Recommended */
.example {}
```

4.1.8 Shorthand Properties

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;
```

4.1.9 0 and Units

Omit unit specification after “0” values, unless required.

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

```
flex: 0px; /* This flex-basis component requires a unit. */
flex: 1 1 0px; /* Not ambiguous without the unit, but needed in IE11. */
margin: 0;
padding: 0;
```

4.1.10 Leading 0s

Always include leading “0”s in values.

Put `0` s in front of values or lengths between -1 and 1.

```
font-size: 0.8em;
```

4.1.11 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;
```

4.1.12 Important Declarations

Avoid using `!important` declarations.

These declarations break the natural cascade of CSS and make it difficult to reason about and compose styles. Use selector specificity to override properties instead.

```
/* Not recommended */
.example {
  font-weight: bold !important;
}
```

```
/* Recommended */
.example {
  font-weight: bold;
}
```

4.1.13 Hacks

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.

4.2 CSS Formatting Rules

4.2.1 Declaration Order

Alphabetize declarations (optional).

Sort declarations consistently within a project. In the absence of tooling to automate and enforce a consistent sort order, consider putting declarations in alphabetical order in order to achieve consistent code in a way that is easy to learn, remember, and manually 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;
```

4.2.2 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;
  }

}
```

4.2.3 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;
}
```

4.2.4 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;
}
```

4.2.5 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;
}
```

4.2.6 Selector and Declaration Separation

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;
}
```

4.2.7 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%;
}
```

4.2.8 CSS Quotation Marks

Use single (`''`) rather than double (`""`) quotation marks for attribute selectors and 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("https://www.google.com/css/maia.css");

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

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

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

4.3 CSS Meta Rules

4.3.1 Section Comments

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.