HTML/CSS Style Guide

https://google.github.io/styleguide/htmlcssguide.xml

HTML / CSS Style

- Maintaining a clear and consistent style for html/css sources is a key requirement for high quality code
  - Promotes readability
  - Simplifies maintenance
  - Ensures an orderly evolution of the code base
- Adopting a single style guide for a team ensures all code is across a project is consistent.

Google HTML/CSS Style Guide

General Style Rules	Protocol
General Formatting Rules	Indentation Capitalization Trailing Whitespace
General Meta Rules	Encoding Comments Action Items
HTML Style Rules	Document Type HTML Validity Semantics Multimedia Fallback Separation of Concerns Entity References Optional Tags type Attributes
HTML Formatting Rules	General Formatting HTML Quotation Marks
CSS Style Rules	CSS Validity ID and Class Naming ID and Class Name Style Type Selectors Shorthand Properties 0 and Units Leading 0s Hexadecimal Notation Prefixes ID and Class Name Delimiters Hacks
CSS Formatting Rules	Declaration Order Block Content Indentation Declaration Stops Property Name Stops Declaration Block Separation Selector and Declaration Separation Rule Separation CSS Quotation Marks
CSS Meta Rules	Section Comments

- A tour through selected recommendations
- Bear in mind for your assignment!

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;
}
```

Formating HTML in Sublime?

stackoverflow

294

There are half a dozen or so ways to format HTML in Sublime. I've tested each of t
plugins (see the writeup I did on my blog for full details), but here's a quick overvie
most popular options:

Reindent command

Pros:

- Ships with Sublime, so no plugin install needed

Cons:

- Doesn't delete extra blank lines
- Can't handle minified HTML, lines with multiple open tags
- Doesn't properly format `<script>` blocks

Tag

Pros:

- Supports ST2/ST3
- Removes extra blank lines
- No binary dependencies

Cons:

- Chokes on PHP tags
- Doesn't handle `<script>` blocks correctly

HTMLTidy

HTML-CSS-JS Prettify

Pros:

- Supports ST2/ST3
- Handles HTML, CSS, JS
- Great integration with Sublime's menus
- Highly customizable
- Per-project settings
- Format on save option

Cons:

- Requires Node.js
- Not great for embedded PHP

Which is best?

HTML-CSS-JS Prettify is the winner in my book. Lots of great features, not much to complain about.

http://stackoverflow.com/questions/8839753/how-do-i-reformat-html-code-using-sublime-text-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;
```

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>
```

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>
```

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.">
```

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!
```

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="//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>
```

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>
```

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>
```

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.

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 {}
```

ID and Class Name Style

▽ 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 {}
```

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;
```

ID and Class Name Delimiters

▽ 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 {}
```

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;
}
```

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%;
}
```

CSS Quotation Marks

▽ 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;
}
```

Parting Words (from google)

“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.”

Complimentary/Alternative Guide

---

http://codeguide.co/

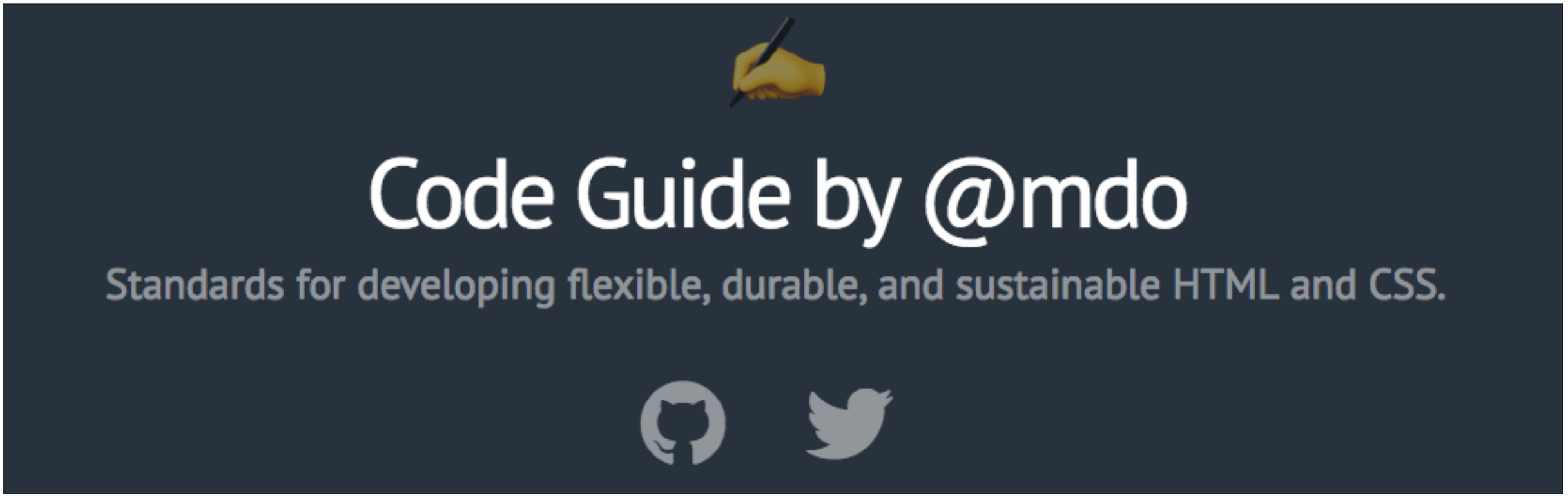

Table of contents

http://codeguide.co/

HTML

- Syntax
- HTML5 doctype
- Language attribute
- Character encoding
- Internet Explorer compatibility mode
- CSS and JavaScript includes
- Practicality over purity
- Attribute order
- Boolean attributes
- Reducing markup
- JavaScript generated markup

CSS

- CSS syntax
- Declaration order
- Don't use @import
- Media query placement
- Prefixed properties
- Rules with single declarations
- Shorthand notation
- Nesting in Less and Sass
- Operators in Less and Sass
- Comments
- Classes
- Selectors
- Organization