Coding Standards
for
NINSHIKI
Version 0.1.0 - Demo #2 Draft

A detailed set of guidelines dictating the best practices, programming styles and conventions that all developers of this project will adhere to.
Please note that these guidelines were heavily influenced by the style guides released by Google[1] under the Creative Commons Licence[2].

Compiled By

The Software Sharks Team
Coetzer MJ
Bester TJ
Orrie O
Bekker L
Lew J
Matodzi MC

Department of Computer Science
The University of Pretoria

For

Bramhope International School of Inovation

[1] http://google.github.io/styleguide/
[2] https://creativecommons.org/licenses/by/3.0/

COS 301 Team: Software Sharks

Department of Computer Science, University of Pretoria

https://github.com/OrishaOrrie/SoftwareSharks

This coding standards document was drafted under the supervision of involved lecturers according to the assessment guidelines of the final year Computer Science module: COS 301 - Software Engineering, presented by the Department of Computer Science in the faculty of Engineering, Built Environment and Information Technology at the University of Pretoria during the first semester of the year 2018.

First release, April 2018

Contents

1 Introduction

2 HTML/CSS

2.1 General

2.1.1 General Style Rules

Protocol

- Never omit the protocol when embedding a resource, image or script.
- Always make use of the HTTPS protocol for embedding resources, images or scripts when available.

```
<!-- Recommended -->
<script src="https://example.com/example/someJava.min.js"></script>
```

2.1.2 General Formatting Rules

Indentation

- Indent with two spaces.
- Do not make use of tabs or various spacing other than the above for indentation.

```
<!-- Recommended -->
<ul>
␣␣<li>Item 1
␣␣<li>Random Item 2
</ul>
```

Capitalisation

- All code has to be in lower-case with the exception of strings.

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

Trailing White Space

- Ensure that there are no trailing white spaces in any line.

2.1.3 General Meta Rules

Encoding

- Make use of UTF-8 encoding that does not include 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.

Comments

- Where ever possible explain code as needed.
- Use comments to explain code: What does it cover, what purpose does it serve, why is respective solution used or preferred?

Action Items

- 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: remove optional tags -->
<ul>
  <li>Tag 1</li>
  <li>Tag 2 (Optional)</li>
</ul>
```

2.2 HTML

2.2.1 HTML Style Rules

Document Type

- Use HTML5.
- HTML5 (HTML syntax) is preferred for all HTML documents: `<!DOCTYPE html>`.
- Do not close void elements, i.e. write `<br>`, not `<br />`.

HTML Validity

- Use valid HTML where possible according to the W3C[3].
- Use valid HTML code unless that is not possible due to otherwise unattainable performance goals regarding file size.

Semantics

- 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.

Multimedia Fall-back

- For multimedia (such as images, videos and animated objects via `<canvas>`) make sure to provide meaningful alternative text via `alt="Descriptive alt text"` to enable global accessibility.

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

Separation of Concerns

- Separate structure (HTML), presentation (CSS) and behaviour (Scripts) into their respective files and file-types.

Entity References

- Do not make use of entity references except for special HTML characters (< and &).

[3] https://validator.w3.org/nu/

`type` Attributes

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

```
<!-- Recommended -->
<link rel="stylesheet" href="https://www.google.com/css/maia.css">
<script src="https://www.google.com/js/gweb/analytics/autotrack.js"></script>
```

2.2.2 HTML Formatting Rules

General Formatting

- Use a new line for every block, list or table element.
- Indent every such child element.

HTML Line-Wrapping

- It is optional to break long lines in order to improve readability.
- If lines are broken then the continuation line should be indented at least 4 additional spaces from the original line.

HTML Quotation Marks

- When quoting attribute values, make use of double quotation marks instead of single quotation marks.

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

2.3 CSS

2.3.1 CSS Style Rules

CSS Validity

- Use valid CSS where possible according to the W3C[4].

ID and Class Naming

- Use meaningful or generic ID and class names that clearly reflect the purpose of the element instead of presentational or cryptic names.

```
/* 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.

Type Selectors

- Avoid qualifying ID and class names with type selectors.

Shorthand Properties

- Use shorthand properties where possible for code efficiency and understandability.

Leading 0s

- Omit leading "0"s in values or lengths between -1 and 1.

```
/* Recommended */
font-size: .8em;
```

[4] https://jigsaw.w3.org/css-validator/

Hexadecimal Notation

- Use 3 character hexadecimal notation where possible.

```
/* Recommended */
color: #ebc; /* Instead of #eebbcc */
```

ID and Class Name Delimiters

- Separate words in ID and class names by a hyphen only.

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

2.3.2 CSS Formatting Rules

Declaration Order

- Alphabetize declarations in order to achieve consistent code in a way that is easy to remember and maintain.
- Ignore vendor-specific prefixes.

Block Content Indentation

- Indent all block content so to reflect hierarchy and improve understanding.

Declaration Stops

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

Property Name Stops

- Use a space after a property name's colon but not before.

Declaration Block Separation

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

```
/* Recommended */
#video {
  margin-top: 1em;
}
```

Selector and Declaration Separation

- Separate selectors and declarations by new lines.

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

Rule Separation

- Separate rules by a blank line.

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()`).

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

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

2.3.3 CSS Meta Rules

Section Comments

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

```
/* Recommended */
/* Header */

#adw-header {}

/* Footer */

#adw-footer {}
```

3 JavaScript

3.1 Source File Basics

3.1.1 File Name

- File names must be all lower-case and may include underscores (_) or dashes (-), but no additional punctuation.
- Follow the convention that your project uses.
- File name extensions must be '`.js`'.

3.1.2 File Encoding

- All source files must be encoded in UTF-8.

3.1.3 Special Characters

Whitespace Characters

- Aside from the line terminator sequence, the ASCII horizontal space character (0x20) is the only whitespace character that appears anywhere in a source file.
- All other whitespace characters in string literals are escaped.
- Tab characters are not used for indentation.

Special escape sequences

- Use special escape sequences: `" \', \", \\, \b, \f, \n, \r, \t, \v "`.
- Do not use numeric or octal escapes.

3.2 Source File Structure

A source file consists of, in order:

1. Licence or copyright information if available.
2. `// @fileoverview` JSDoc.
3. File implementation.

Exactly one blank line separates each section that is present.

3.3 Formatting

3.3.1 Braces

Braces are Used for all Control Structures

- Braces are required for all control structures.
- The first statement of a non-empty block must begin on its own line.

Non-empty blocks: K&R style

Braces follow the Kernighan and Ritchie style ("Egyptian brackets") for non-empty blocks and block-like constructs:

- No line break before the opening brace.
- Line break after the opening brace.
- Line break before the closing brace.
- Line break after the closing brace if that brace terminates a statement or the body of a function or class statement, or a class method.

```
class InnerClass {
  constructor() {}

  /** @param {number} foo */
  method(foo) {
    if (condition(foo)) {
      try {
        // Note: this might fail.
        something();
      } catch (err) {
        recover();
      }
    }
  }
}
```

Empty blocks: may be concise

- An empty block or block-like construct may be closed immediately after it is opened, with no characters, space, or line break in between.

3.3.2 Block indentation: +2 spaces

Array literals: optionally "block-like"

4 Python

4.1 Python Style Rules

Semi-colons

- Do not terminate your lines with semi-colons and do not use semi-colons to put two commands on the same line.

Line length

- Maximum line length is 80 characters.
- Exceptions:
  - Long import statements.
  - URLs in comments.
- Do not use backslash line continuation.
- Make use of Python's implicit line joining inside parentheses, brackets and braces. If necessary, you can add an extra pair of parentheses around an expression.
- When a literal string won't fit on a single line, use parentheses for implicit line joining.
- Within comments, put long URLs on their own line if necessary.

```
Yes:  # See details at
      # https://www.example.com/us/developer/documentation/api/content/v2.0/csu
```

Parenthesis

- Use parentheses sparingly. Do not use them in return statements or conditional statements unless using parentheses for implied line continuation. It is however fine to use parentheses around tuples.

```
Yes: if foo:
         bar()
     while x:
         x = bar()
     if x and y:
         bar()
     if not x:
         bar()
     return foo
     for (x, y) in dict.items(): ...
```

```
No:  if (x):
         bar()
     if not(x):
         bar()
     return (foo)
```

Indentation

- Indent your code blocks with 4 spaces.
- Never use tabs or mix tabs and spaces. In cases of implied line continuation, you should align wrapped elements either vertically, as per the examples in the line length section; or using a hanging indent of 4 spaces, in which case there should be no argument on the first line.

```
Yes:   # Aligned with opening delimiter
       foo = long_function_name(var_one, var_two,
                                var_three, var_four)

       # Aligned with opening delimiter in a dictionary
       foo = {
           long_dictionary_key: value1 +
                                value2,
           ...
       }

       # 4-space hanging indent; nothing on first line
       foo = long_function_name(
           var_one, var_two, var_three,
           var_four)

       # 4-space hanging indent in a dictionary
       foo = {
           long_dictionary_key:
               long_dictionary_value,
           ...
       }
```

Blank Lines

- Two blank lines between top-level definitions, one blank line between method definitions.
- Two blank lines between top-level definitions, be they function or class definitions.

- One blank line between method definitions and between the class line and the first method. Use single blank lines as you judge appropriate within functions or methods.

Whitespace

- Follow standard typographic rules for the use of spaces around punctuation.
- No whitespace inside parentheses, brackets or braces.
- No whitespace before a comma, semicolon, or colon. Do use whitespace after a comma, semicolon, or colon except at the end of the line.

```
Yes: if x == 4:
         print x, y
     x, y = y, x
```

- No whitespace before the open paren/bracket that starts an argument list, indexing or slicing.
- Surround binary operators with a single space on either side for assignment (=), comparisons (==, <, >, !=, <>, <=, >=, in, not in, is, is not), and Boolean (and, or, not). Use your better judgement for the insertion of spaces around arithmetic operators but always be consistent about white-space on either side of a binary operator.
- Don't use spaces around the '=' sign when used to indicate a keyword argument or a default parameter value.

```
Yes: def complex(real, image =0.0): return magic(r=real, i=image)
```

- Don't use spaces to vertically align tokens on consecutive lines, since it becomes a maintenance burden (applies to :, =, etc.):

```
Yes:
  foo = 1000  # comment
  long_name = 2  # comment that should not be aligned

  dictionary = {
      'foo': 1,
      'long_name': 2,
  }
```