Languages used: HTML, CSS, Javascript, React

# HTML/CSS Guide

* Always use HTTPS for images and other media files unless the respective files are not available over HTTPS
* indentation
  + indent by 2 spaces at a time
  + don’t use tabs or mix tabs and spaces for indentation
* capitalization
  + only use lowercase
    - applies to HTML element names, attributes, attribute values, CSS selectors, properties, and property values (with the exception of strings)
    - good example: color: #e5e535;
    - bad example: color: #E5E5E5;
* remove trailing white spaces
  + trailing white spaces are unnecessary and can complicate code
  + good example: yes please.
  + bad example: What?\_
* use UTF-8 encoding
  + make sure your editor uses UTF-8 as character encoding, without a byte order mark
  + specify the encoding in HTML templates and documents via <code>&lt;meta charset=”utf-8”&gt;</code>
* explain comments as needed, where possible
  + use comments to explain code: what does it cover, what purpose does it serve, why is the specific solution used or preferred?
* use HTML5
  + HTML5 (HTML syntax) is preferred for all HTML documents
* general formatting
  + use a new line for every block, list, or table element, and indent every such child element
* 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 at least 4 additional spaces from the original line to distinguish wrapped attributes from child elements
* 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
* leading 0s
  + always include leading “0”s in values
  + put 0 in front of values or lengths between -1 and 1
* hexadecimal notation
  + use 3 character hexadecimal notation where possible
  + for color values that permit it, 3 character hexadecimal notation is shorter and more succinct
  + good example: color: #ebc;
  + bad example: color: #eebbcc;

# CSS formatting rules

* + alphabetize declarations
    - put declarations in alphabetical order in order to achieve consistent code in a way that is easy to remember and maintain
  + indent all block content
    - indent all block content that is rules within rules as well as declarations, so to reflect hierarchy and improve understanding
  + declaration stops
    - use a semicolon after every declaration
      * end every declaration with a semicolon for consistency and extensibility reasons
    - good example

.test {

display: block;

height: 100px;

}

* + - bad example

.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
    - good example:

h3 {

font-weight: bold;

}

* + - bad example:

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
  + selector and declaration separation
    - separate selectors and declarations by new lines
    - always start a new line for each selector and declaration
  + rule separation
    - separate rules by new lines
    - always put a blank line (two line breaks) between rules
  + CSS quotation marks
    - use single (‘ ‘) rather than double quotation marks for attribute selectors and property values
    - do not use quotation marks in URI values

# Javascript Style Guide

* Use spaces, not tabs
* Aside from the line terminator sequence, the ASCII horizontal space character (0x20) is the only whitespace character that appears anywhere in a source file. This implies that… Tab characters are not used for indentation.

Example: badfunction foo() {∙∙∙∙let name;}// badfunction bar() {∙let name;} // goodfunction baz() {∙∙let name;}

* Semicolons ARE required
* Every statement must be terminated with a semicolon. Relying on automatic semicolon insertion is forbidden.

// badlet luke = {}let leia = {}[luke, leia].forEach(jedi => jedi.father = 'vader')

// goodlet luke = {};let leia = {};[luke, leia].forEach((jedi) => { jedi.father = 'vader';});

* Horizontal alignment is discouraged
* This practice is permitted, but it is generally discouraged by Google Style. It is not even required to maintain horizontal alignment in places where it was already used.

Horizontal alignment is the practice of adding a variable number of additional spaces in your code, to make certain tokens appear directly below certain other tokens on previous lines.

// bad{ tiny: 42, longer: 435, };

// good{ tiny: 42, longer: 435,};

* Don’t use var anymore
* Declare all local variables with either const or let. Use const by default, unless a variable needs to be reassigned. The var keyword must not be used.

// bad var example = 42;

// good let example = 42;

* Arrow functions are preferred
* Arrow functions provide a concise syntax and fix a number of difficulties with this. Prefer arrow functions over the function keyword, particularly for nested functions
* // bad[1, 2, 3].map(function (x) { const y = x + 1; return x \* y;});
* // good[1, 2, 3].map((x) => { const y = x + 1; return x \* y;});
* Use template strings instead of concatenation
* Use template strings (delimited with `) over complex string concatenation, particularly if multiple string literals are involved. Template strings may span multiple lines.
* // badfunction sayHi(name) { return 'How are you, ' + name + '?';}// badfunction sayHi(name) { return ['How are you, ', name, '?'].join();}// badfunction sayHi(name) { return `How are you, ${ name }?`;}// goodfunction sayHi(name) { return `How are you, ${name}?`;}
* Don’t use line continuations for long strings
* Do not use line continuations (that is, ending a line inside a string literal with a backslash) in either ordinary or template string literals. Even though ES5 allows this, it can lead to tricky errors if any trailing whitespace comes after the slash, and is less obvious to readers.
* // bad (sorry, this doesn't show up well on mobile)const longString = 'This is a very long string that \ far exceeds the 80 column limit. It unfortunately \ contains long stretches of spaces due to how the \ continued lines are indented.';
* // goodconst longString = 'This is a very long string that ' + 'far exceeds the 80 column limit. It does not contain ' + 'long stretches of spaces since the concatenated ' + 'strings are cleaner.';
* “for… of” is the preferred type of ‘for loop’
* With ES6, the language now has three different kinds of for loops. All may be used, though for-of loops should be preferred when possible.
* Don’t use eval()
* Do not use eval or the Function(...string) constructor (except for code loaders). These features are potentially dangerous and simply do not work in CSP environments.
* The MDN page for eval() even has a section called “Don’t use eval!”
* // badlet obj = { a: 20, b: 30 };let propName = getPropName(); // returns "a" or "b"eval( 'var result = obj.' + propName );
* // goodlet obj = { a: 20, b: 30 };let propName = getPropName(); // returns "a" or "b"let result = obj[ propName ]; // obj[ "a" ] is the same as obj.a
* Constants should be named in ALL\_UPPERCASE separated by underscores
* Constant names use CONSTANT\_CASE: all uppercase letters, with words separated by underscores.
* // badconst number = 5;
* // goodconst NUMBER = 5;
* One variable per declaration
* Every local variable declaration declares only one variable: declarations such as let a = 1, b = 2; are not used.
* // badlet a = 1, b = 2, c = 3;
* // goodlet a = 1;let b = 2;let c = 3;
* Use single quotes, not double quotes
* Ordinary string literals are delimited with single quotes ('), rather than double quotes (").
* // badlet directive = "No identification of self or mission."
* // badlet saying = 'Say it ain\u0027t so.';
* // goodlet directive = 'No identification of self or mission.';
* // goodlet saying = `Say it ain't so`;

# 

# React Style Guide

* Only include one React component per file
  + multiple Stateless, or Pure, Components are allowed per file
* Do not use React.createElement unless you’re initializing the app from a file that is not JSX
* If you have internal state and/or refs, prefer class extends React.Component over React.createClass
* if you don’t have state or refs, prefer normal functions (not arrow functions) over classes
* do not use mixins
* naming
  + Use .js extension for React components
  + Use PascalCase for filenames. E.g., ReservationCard.js
  + Use the filename as the component name
    - good ex: import Footer from './Footer';
    - bad ex: import Footer from './Footer/Footer';
  + Do not use displayName for naming components. Instead, name the component by reference
  + Always use camelCase for prop names
* Always use double quotes (") for JSX attributes, but single quotes (') for all other JS
* Always include a single space in your self-closing tag

HTML/CSS Style Guide Reference: <https://google.github.io/styleguide/htmlcssguide.html>

Javascript Style Guide Reference: <https://google.github.io/styleguide/jsguide.html#features-array-literals>

React Style Guide Reference:

<https://airbnb.io/javascript/react/>