AlarmBuddy Style Guide

Team Members:
Joshua Ringling: ring7885@stthomas.edu
Hannah Balut: balu2673@stthomas.edu
Alex Notch: notc6377@stthomas.edu
Mitzi Bustamante Chicaiza: bust6340@stthomas.edu

Background:

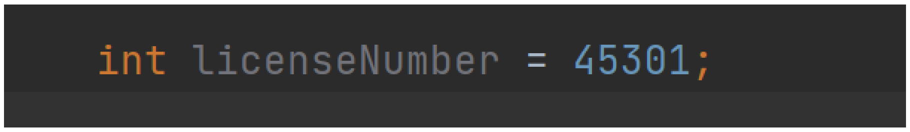
This style guide is to be used as a reference for all work being done related to the AlarmBuddy project. The attached guides were heavily inspired by style guidelines provided by Google under the CC-By 3.0 License, which encourages the sharing of these and the original source documents. The original style guides provided by Google can be found at https://google.github.io/styleguide/

Documentation Guidelines:

- Any services requiring accounts being used should have detailed instructions on how to create and login to an account.
- Any software or tools used in production such as IDEs should have detailed instructions on downloading and installing the software such that settings are the same used while making the project.

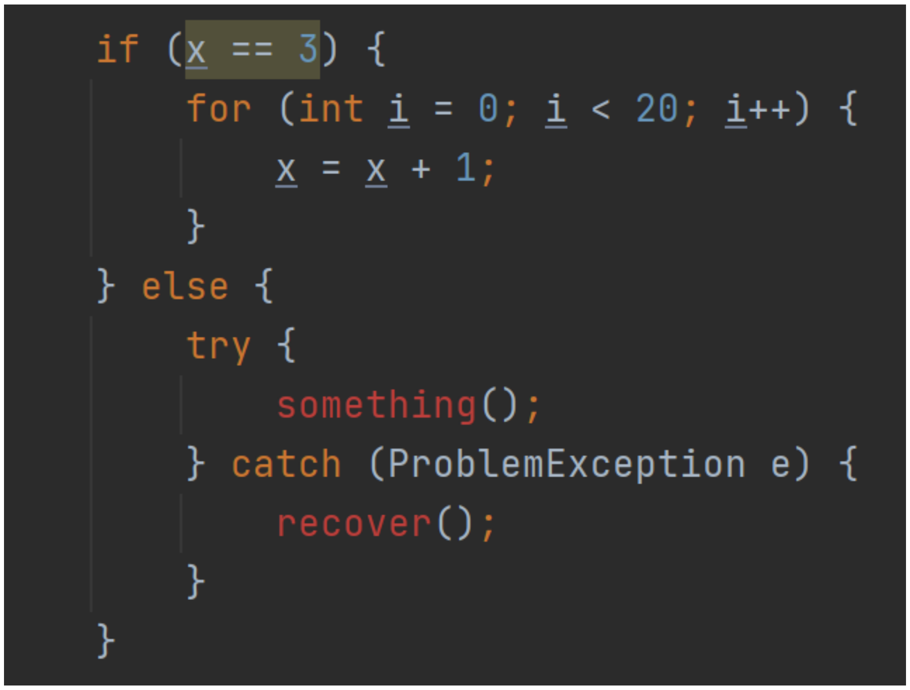
General Rules:

- Variables start with lower case letters and use context appropriate names. Use camel-case for variables that use multiple words.



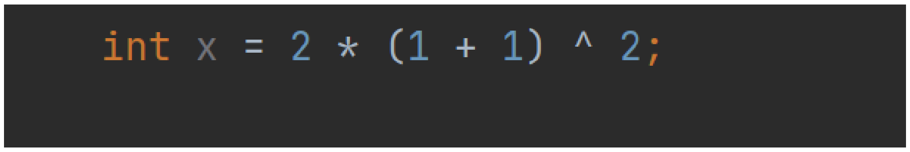
```
int licenseNumber = 45301;
```

- Indent using tab to properly show scope. Examples: if statements, for loops, while loops, and functions.
- Use curly braces on all if-else statements, loops, and functions even when they only contain one line.
- Curly braces follow the Kernighan and Ritchie style (or “Egyptian brackets”) for code blocks:
    - 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 *only if* that brace terminates the current body of a method, constructor, or named class. For example, there is no line break after a brace if an else is being used.



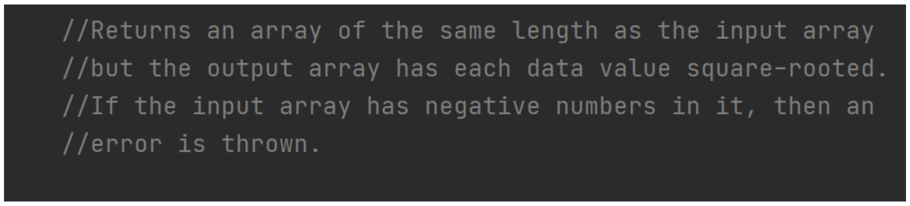
```
if (x == 3) {
    for (int i = 0; i < 20; i++) {
        x = x + 1;
    }
} else {
    try {
        something();
    } catch (ProblemException e) {
        recover();
    }
}
```

- In mathematical operations, have spaces between all operators and operands *except* for parenthesis and use parentheses to show clear mathematical steps.



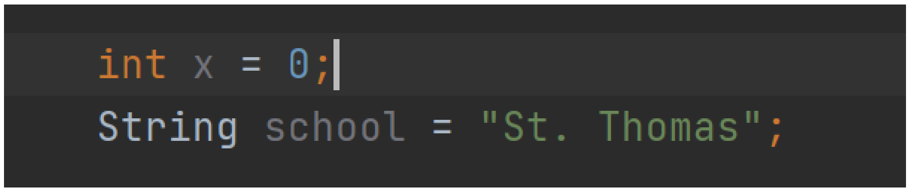
```
int x = 2 * (1 + 1) ^ 2;
```

- Comment the use of functions as well as any special exceptions that may happen.
- Avoid making comments longer than 120 characters long. Try to wrap excessively long sections onto the next line.



```
//Returns an array of the same length as the input array
//but the output array has each data value square-rooted.
//If the input array has negative numbers in it, then an
//error is thrown.
```

- Try to initialize variables when declaring them. However, this is not always possible.



```
int x = 0;
String school = "St. Thomas";
```

HTML/CSS Style Guide

Joshua Ringling: ring7885@stthomas.edu

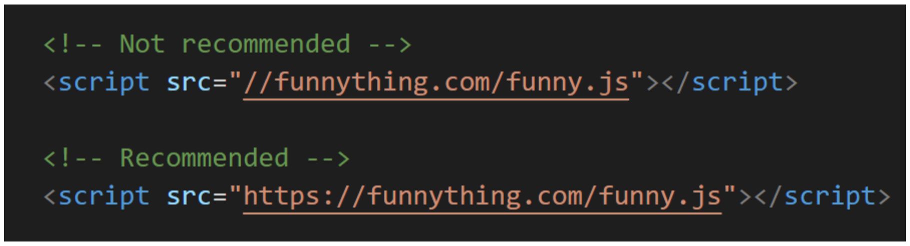
Background:

This document defines the formatting and style rules for HTML and CSS. It applies to raw, working files that use HTML and CSS coding.

General Rules:

Protocol:

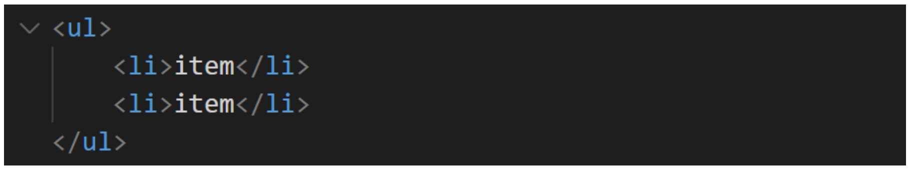
Use HTTPS for embedded resources where possible. Always use HTTPS for images, style sheets, and scripts except for when not available over HTTPS.



```html
<!-- Not recommended -->
<script src="//funnything.com/funny.js"></script>

<!-- Recommended -->
<script src="https://funnything.com/funny.js"></script>
```

Indentation:

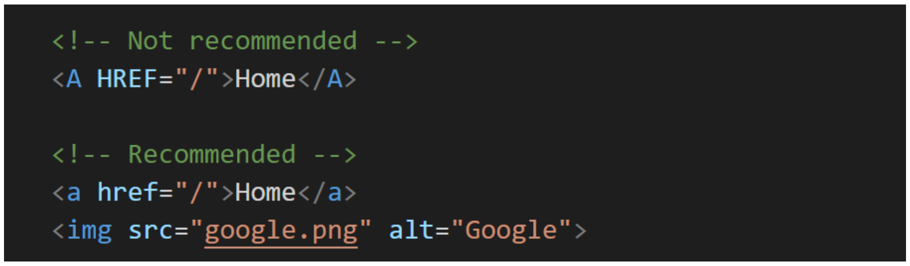
Indent using a single tab. Don't mix tabs and spaces for indentation.



```html
<ul>
	<li>item</li>
	<li>item</li>
</ul>
```

Capitalization:

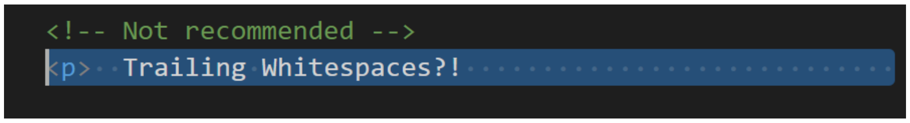
Use lowercase.
All HTML and CSS code should be lowercase including element names, attributes, attribute values, CSS selectors, properties, and property values except for strings, comments, and paragraph text.



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

<!-- Recommended -->
<a href="/">Home</a>
<img src="google.png" alt="Google">
```

Trailing Whitespace:

Remove any trailing white spaces.
Trailing white spaces are unnecessary and can complicate code.



```
<!-- Not recommended -->
<p>  Trailing Whitespaces?!                         
```

Encoding:

Use UTF-8.

Make sure your IDE editor is using UTF-8 as character encoding.

Specify the encoding in HTML templates and documents via

<meta charset="utf-8">

Comments:

Explain code as needed when possible.

Difficult sections of code should be documented explaining its function.

HTML Style Rules:

Document Type:

Use HTML 5

HTML syntax is to be used for all HTML documents: <!DOCTYPE html>.

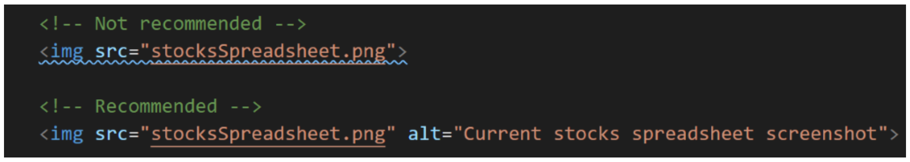
Semantics:

Use HTML elements for what they have been created for. For example, use heading elements for headings, p elements for paragraphs, and a elements for anchors.

Multimedia Fallback

Provide alternative contents for any multimedia.

Providing alternative contents is important for accessibility: A blind user using a reading application will have no cue without an alt text.



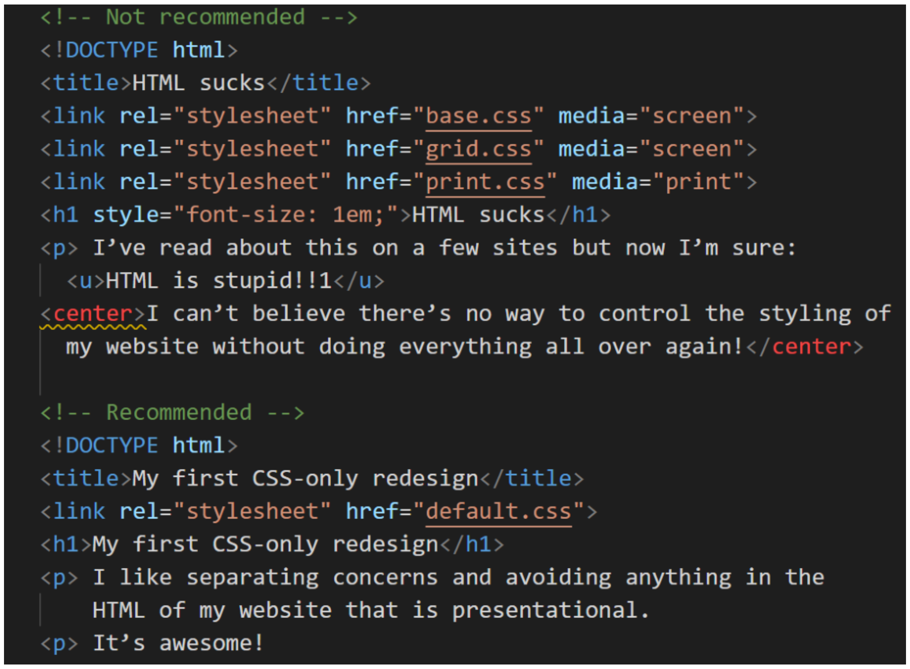
```
<!-- Not recommended -->
<img src="stocksSpreadsheet.png">

<!-- Recommended -->
<img src="stocksSpreadsheet.png" alt="Current stocks spreadsheet screenshot">
```

Separation of Code:

Strictly keep structure (markup), presentation (styling), and behavior (scripting) apart, and try to keep the interaction between them to an absolute minimum.

For example, make sure documents and templates contain only HTML that is solely serving structural purposes. Move everything used for presentation into style sheets, and everything behavioral into scripts.

In addition, keep contact as small as possible by linking as few style sheets and scripts to a single HTML document as possible.



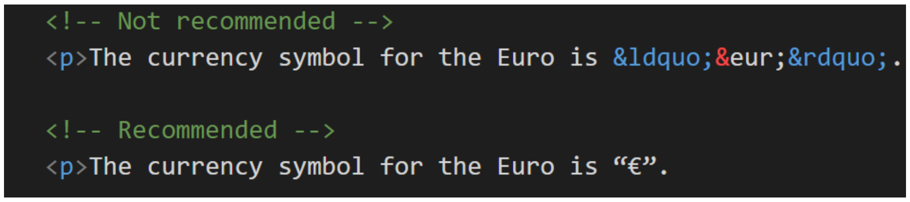
```html
<!-- Not recommended -->
<!DOCTYPE html>
<title>HTML sucks</title>
<link rel="stylesheet" href="base.css" media="screen">
<link rel="stylesheet" href="grid.css" media="screen">
<link rel="stylesheet" href="print.css" media="print">
<h1 style="font-size: 1em;">HTML sucks</h1>
<p> I’ve read about 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 like separating concerns and avoiding anything in the
    HTML of my website that is presentational.
<p> It’s awesome!
```

Entity References:

Do not use any entity references.

UTF-8 is used among all teams, files, and editors. There is no need to use entity references such as &mdash;, &rdquo;, or &eur.

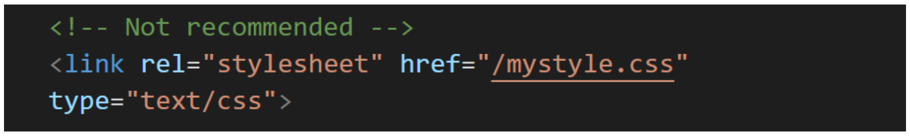
The only exceptions to this rule are characters with special meaning in HTML like < and & as well as control characters like no-break spaces.



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

<!-- Recommended -->
<p>The currency symbol for the Euro is “€”.
```

Type Attributes:

Do not include type attributes for style sheets and scripts as HTML5 implies them automatically.



```html
<!-- Not recommended -->
<link rel="stylesheet" href="/mystyle.css"
type="text/css">
```

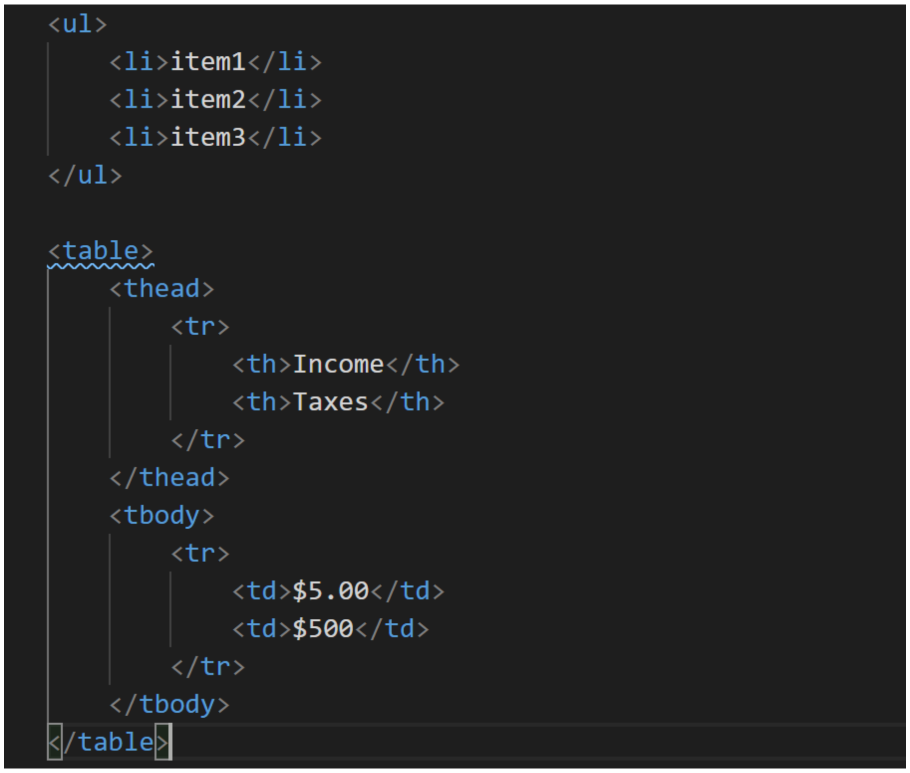
HTML Formatting Rules:

General Formatting:

Use a new line for every block, list, or table element, and indent every such child element.

Place every block, list, or table element on a new line.

Indent any element that is the child element of a block, list, or table element.



```html
<ul>
    <li>item1</li>
    <li>item2</li>
    <li>item3</li>
</ul>

<table>
    <thead>
        <tr>
            <th>Income</th>
            <th>Taxes</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td>$5.00</td>
            <td>$500</td>
        </tr>
    </tbody>
</table>
```

HTML Line-Wrapping:

(optional) Break long lines.
When possible, consider wrapping long lines if it significantly improves readability.
When line-wrapping, each continuation line should be indented by 1 tab from the original line.

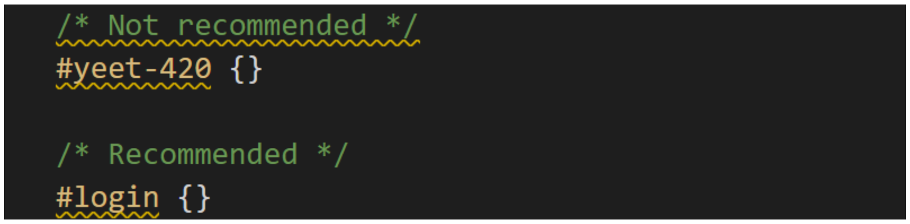
HTML Quotation Marks

When quoting attribute values, use double quotation marks (“”).

CSS Style Rules:

ID and Class Names:

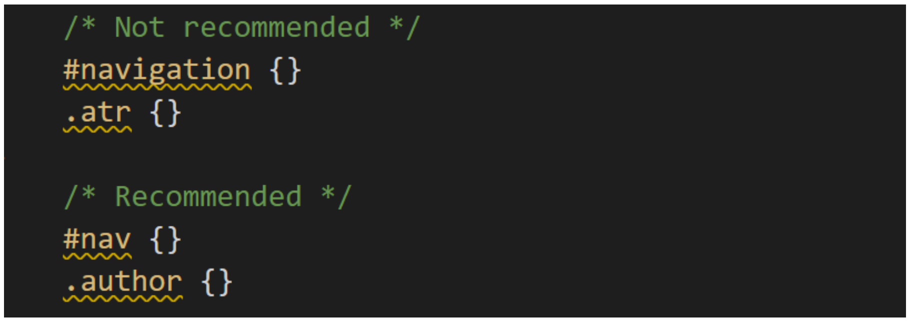
Use meaningful or generic ID and class names.
Don’t use cryptic names. Always use ID and class names that reflect the purpose of the element or ones that are generic.
Meaningful ID and class names are preferred. Generic names are a fallback for elements that have no meaning different from their siblings.



```css
/* Not recommended */
#yeet-420 {}

/* Recommended */
#login {}
```

ID and Class Names Length:

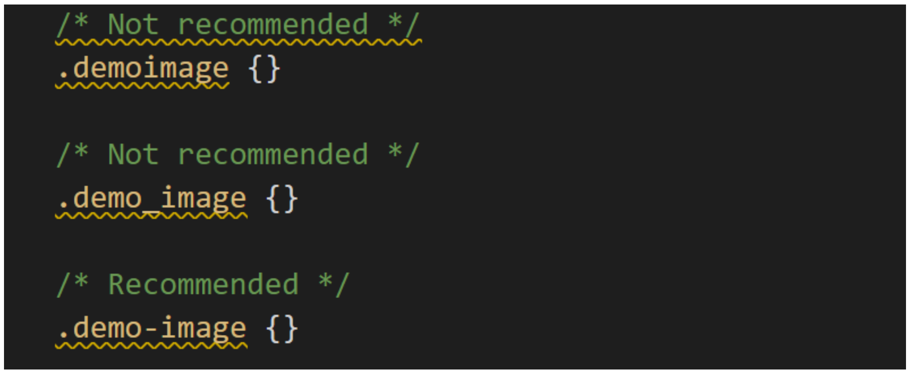
Use ID and class names that are as short as possible but as long as necessary. Try to convey what an ID or class is while being as brief as possible.



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

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

ID and Class Name Delimiters:

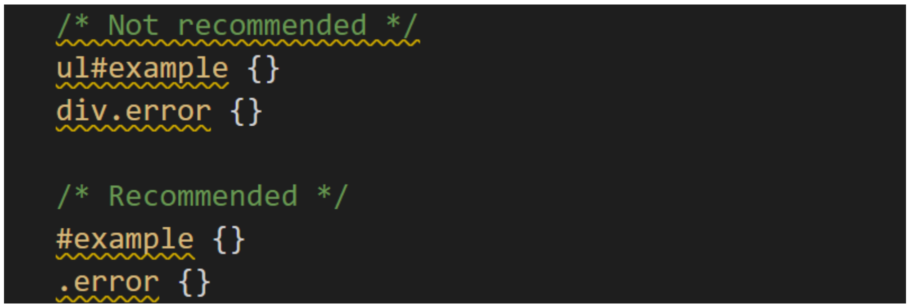
Separate multiple words in ID and class names by a hyphen.



```
/* Not recommended */
.demoimage {}

/* Not recommended */
.demo_image {}

/* Recommended */
.demo-image {}
```

Type Selectors:

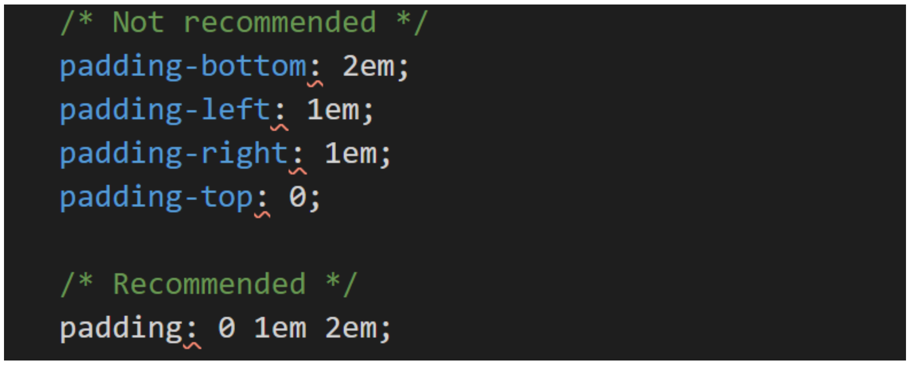
Avoid using ID and class names with type selectors.
Using unnecessary ancestor selectors hinders performance and load times.



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

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

Shorthand Properties:

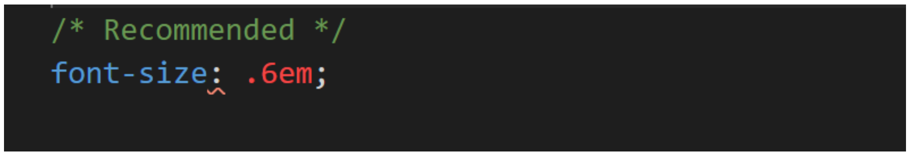
(optional) Use shorthand properties when possible.
Using shorthand properties is useful for code efficiency and understandability.



```css
/* Not recommended */
padding-bottom: 2em;
padding-left: 1em;
padding-right: 1em;
padding-top: 0;

/* Recommended */
padding: 0 1em 2em;
```

Leading 0s:

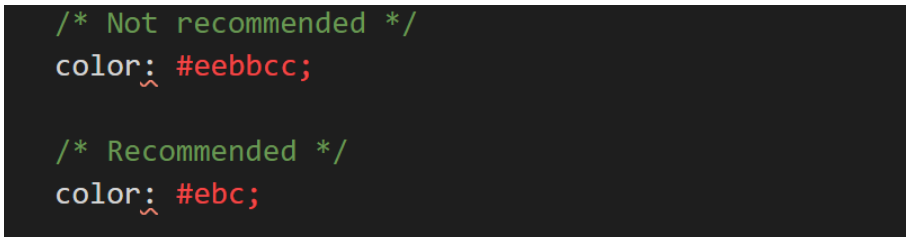
Omit leading “0”s in values that are between -1 and 1.



```css
/* Recommended */
font-size: .6em;
```

Hexadecimal Notation:

Use 3 character hexadecimal notation where possible.
For the majority of color values, 3 character hexadecimal notation is shorter and more succinct.

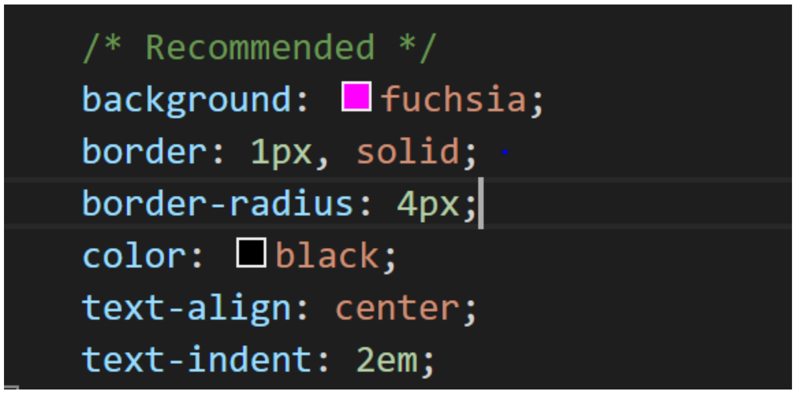


```css
/* Not recommended */
color: #eebbcc;

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

CSS Format Rules:

Declaration Order:

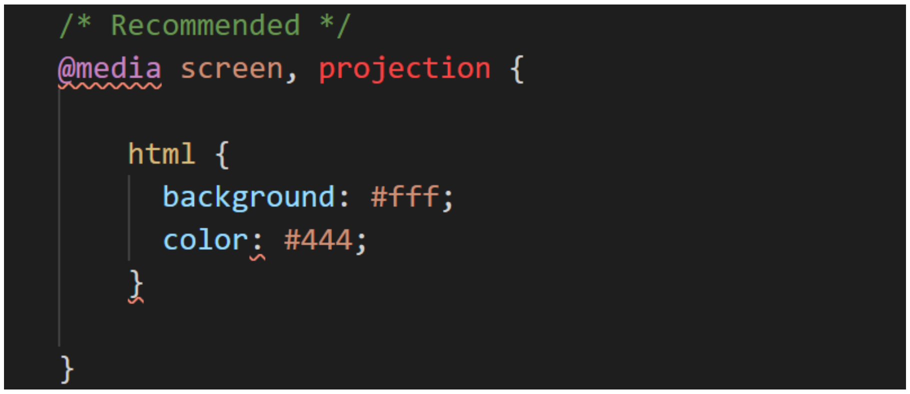
Alphabetize declarations.
Put declarations in alphabetical order to achieve consistent code that is easy to remember and maintain.



```css
/* Recommended */
background: fuchsia;
border: 1px, solid;
border-radius: 4px;
color: black;
text-align: center;
text-indent: 2em;
```

Block Indentation:

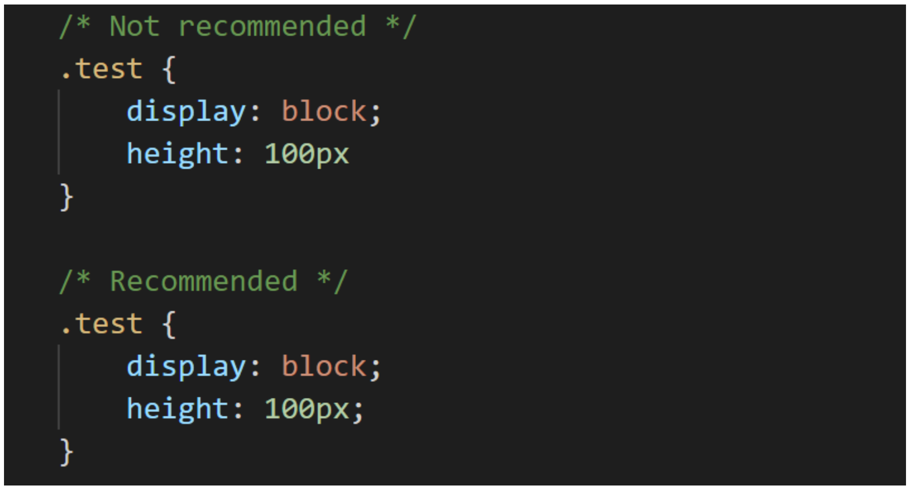
Indent all block content such as rules within rules as well as their declarations to properly reflect scope.



```css
/* Recommended */
@media screen, projection {

    html {
      background: #fff;
      color: #444;
    }

}
```

Declaration Stops:

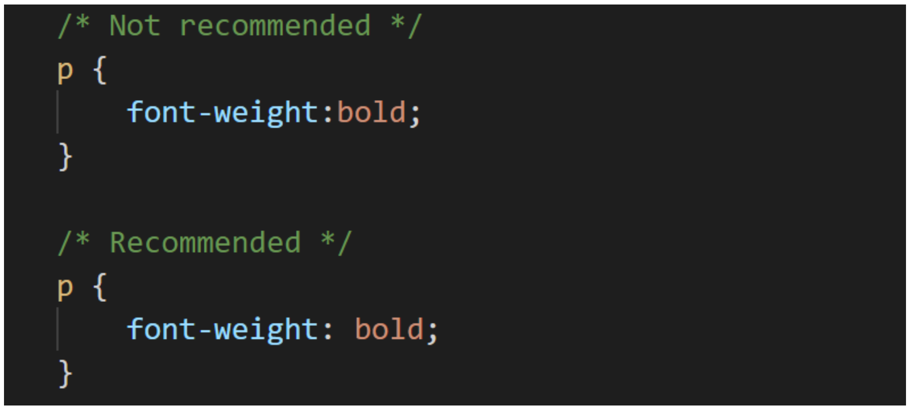
Use a semicolon after every declaration for consistency and clarity.



```css
/* Not recommended */
.test {
    display: block;
    height: 100px
}

/* Recommended */
.test {
    display: block;
    height: 100px;
}
```

Property Name Stops:

Always use a single space after a property name's colon.

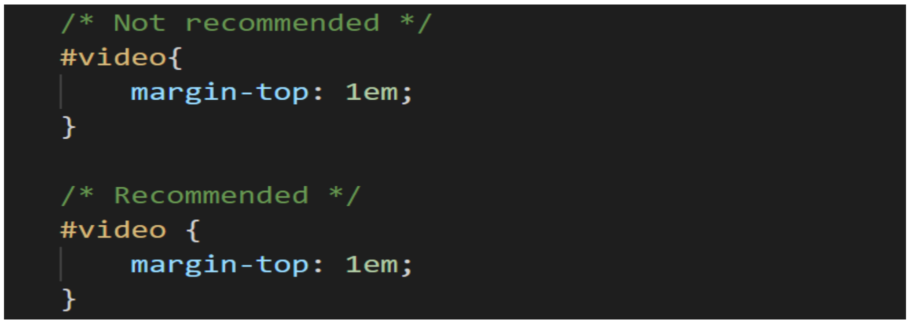


```css
/* Not recommended */
p {
    font-weight:bold;
}

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

Declaration Block Separation:

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.

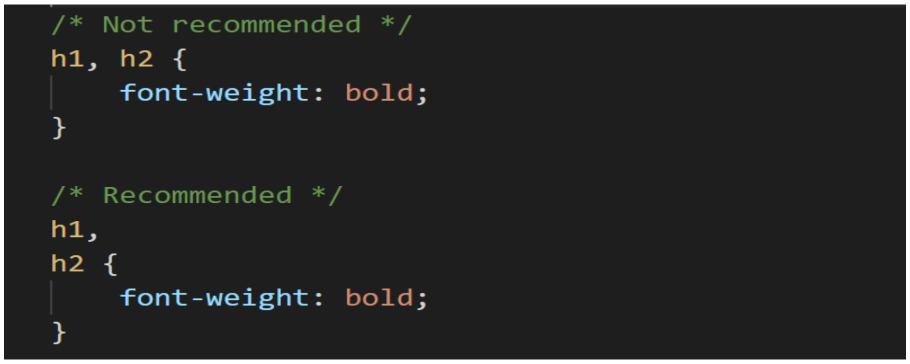


```css
/* Not recommended */
#video{
    margin-top: 1em;
}

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

Selector and Declaration Separation:

Separate selectors and declarations by new lines.

Always start a new line for each new selector and declaration.

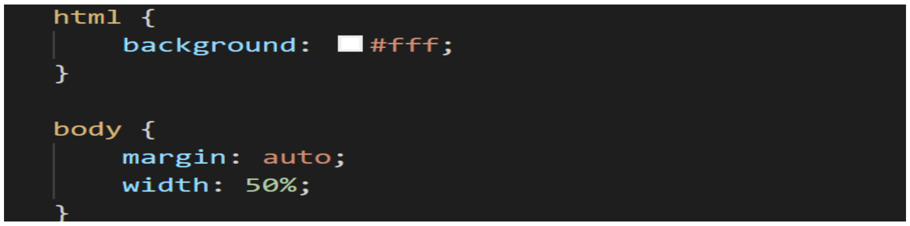


```css
/* Not recommended */
h1, h2 {
    font-weight: bold;
}

/* Recommended */
h1,
h2 {
    font-weight: bold;
}
```

Rule Separation:

Separate rules by new lines.

Always put a blank line between rules.

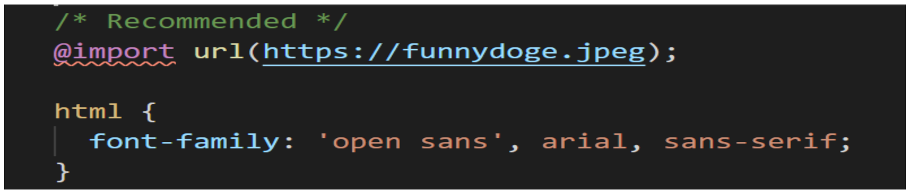


```css
html {
    background: #fff;
}

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

CSS Quotation Marks:

Use single (‘’) quotation marks for attribute selectors and property values.

Do not use quotation marks in URL values.



```css
/* Recommended */
@import url(https://funnydoge.jpeg);

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

JavaScript Style Guide

Joshua Ringling: ring7885@stthomas.edu

Background:

This document defines the formatting and style rules for JavaScript code. It applies to raw, working files that use JavaScript coding.

General Rules:

File Names:

File names must be all lowercase and may include underscores (_) or dashes (-) , but not additional punctuation.

The file's extension must be .js.

File Encoding:

All JavaScript files should be encoded using UTF-8.

Formatting Rules:

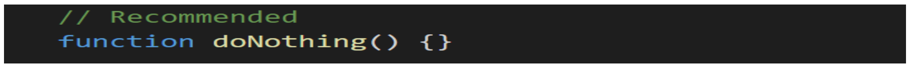
Braces:

Braces are required for all control structures (if, else, for, while, etc.), even if the body contains only a single statement.

All braces should follow the "Egyptian" brackets style mentioned above.

Empty Code Blocks:

An empty code block or block-like construct should be closed immediately if it contains nothing.



```
// Recommended
function doNothing() {}
```

Indentation:

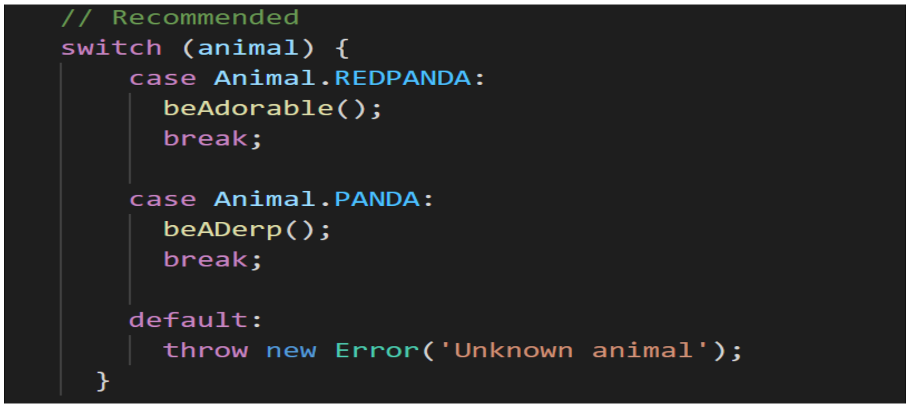
Code blocks such as functions, if statements, or for loops should be indented using a single tab to properly show their scope.

Switch Statements:

Switch statements should be indented like any other block code.

The indentation for a switch statement example can be viewed down below.

All switch statements should have a default case.



```
// Recommended
switch (animal) {
    case Animal.REDPANDA:
      beAdorable();
      break;

    case Animal.PANDA:
      beADerp();
      break;

    default:
      throw new Error('Unknown animal');
  }
```

One Statement Per Line:

There should be only one statement on a line of code that is followed by a line-break.

Each statement must be terminated with a semicolon.

Line Wrapping:

JavaScript code should be limited to 80 characters per line. Any code that would exceed this limit must be line-wrapped.

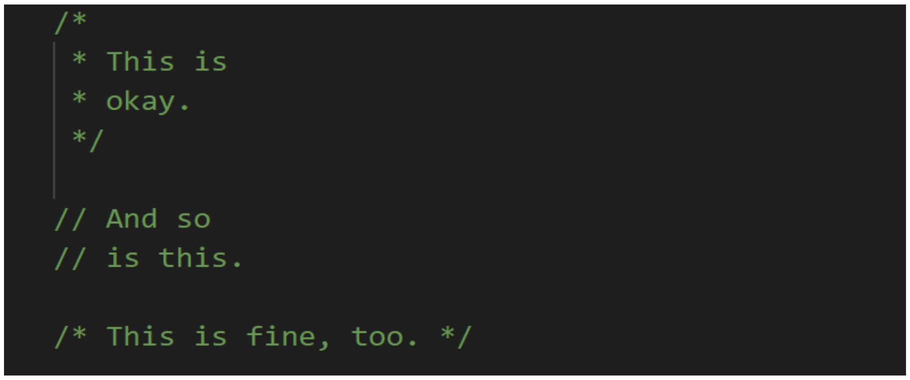
There are no clear-cut rules for line wrapping other than to indent the wrapped code on its next line.

Try to wrap code to produce the best readability possible.

Comments:

Comments should be indented to the same degree as the code they are referring to.

There are many different styles of commenting. All are allowed, but be consistent.



```
/*
 * This is
 * okay.
 */

// And so
// is this.

/* This is fine, too. */
```

Language Features:

Const and Let:

Declare all local variables with const or let. Use const by default, unless a variable needs to be reassigned.

Do not use var to declare variables.

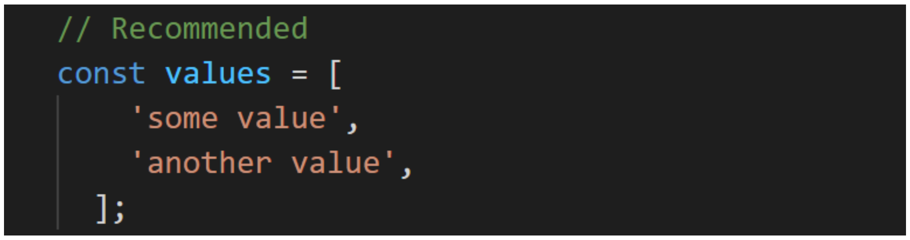
One Declaration At a Time:

Every local variable declaration should declare only one variable.

Do not use declarations such as let a = 0, b = 3;.

Array Literals:

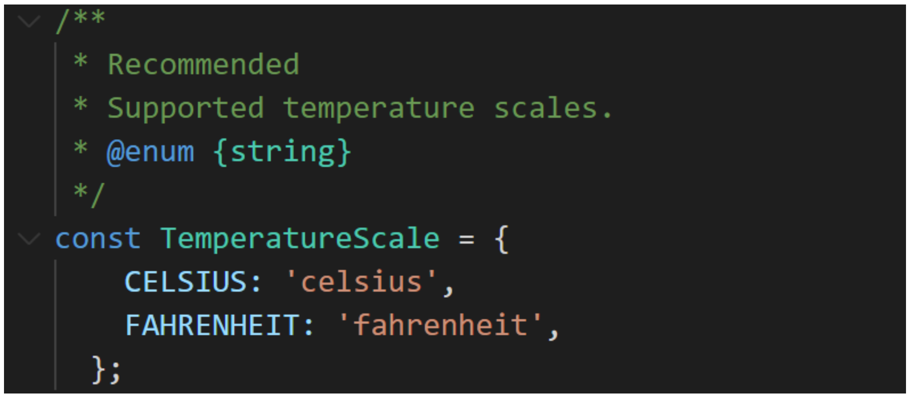
Include a trailing comma whenever there is a line break between the final element and the closing bracket when declaring array values.



```
// Recommended
const values = [
    'some value',
    'another value',
  ];
```

Enums:

Enumerations are defined by adding the @enum annotation to an object literal. Additional properties should not be added to an enum after it is defined. Enums must be constant, and all enum values must be deeply immutable.



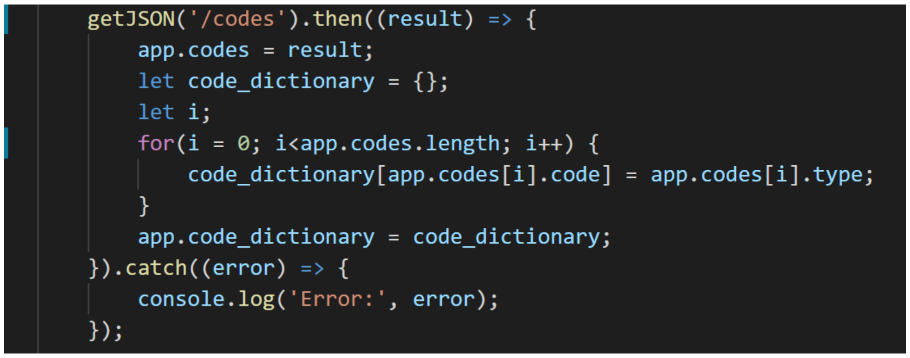
```
/**
 * Recommended
 * Supported temperature scales.
 * @enum {string}
 */
const TemperatureScale = {
    CELSIUS: 'celsius',
    FAHRENHEIT: 'fahrenheit',
  };
```

Overriding toString:

The toString method may be overridden, but must always succeed and never have visible side effects.

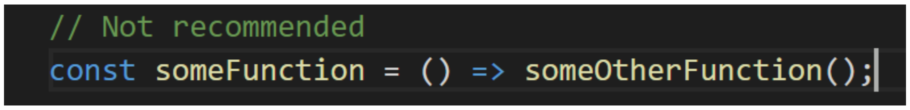
Arrow Functions:

Arrow functions are allowed anywhere an arrow function would make reasonable sense.



```
getJSON('/codes').then((result) => {
    app.codes = result;
    let code_dictionary = {};
    let i;
    for(i = 0; i<app.codes.length; i++) {
        code_dictionary[app.codes[i].code] = app.codes[i].type;
    }
    app.code_dictionary = code_dictionary;
}).catch((error) => {
    console.log('Error:', error);
});
```

Functions with no params and no return value should not use arrow functions.



```
// Not recommended
const someFunction = () => someOtherFunction();
```

Ternary Operators:

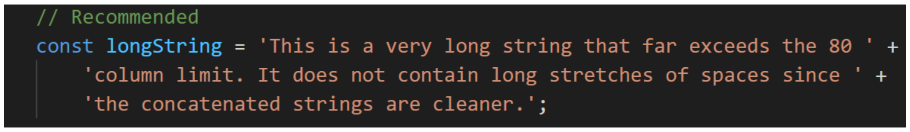
Do not use ternary operators: let i = (age >= 21) ? “Beer” : “Juice”;
Ternary operators make code difficult to interpret upon a glance and are not supported by some out-of-date browsers.

String Literals:

Ordinary String literals are marked with single quotes (‘), rather than double quotes.

String Line Continuations:

Use the + operator to line wrap long String literals.



```
// Recommended
const 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.';
```

This:

Only use this in class constructors and methods or in arrow functions defined within class constructors or methods.

Never use this to refer to a global object.

Equality Checks:

Use identity operators (=== / !==) except when trying to catch either null or undefined.

Constructor Omissions:

Never define a constructor with a new statement without using parentheses ( ) .

Naming Rules:

Camel-Case:

Methods and variables that use multiple words should use camel-case as described above.

Classes and Enums should also use camel-case, but start with a capital letter as well.

	Prose Form	Correct Camel-Case
Variable	“XML HTTP request”	XmlHttpRequest
Variable	“new customer ID”	newCustomerId
Class	“Online Timer App”	OnlineTimerApp

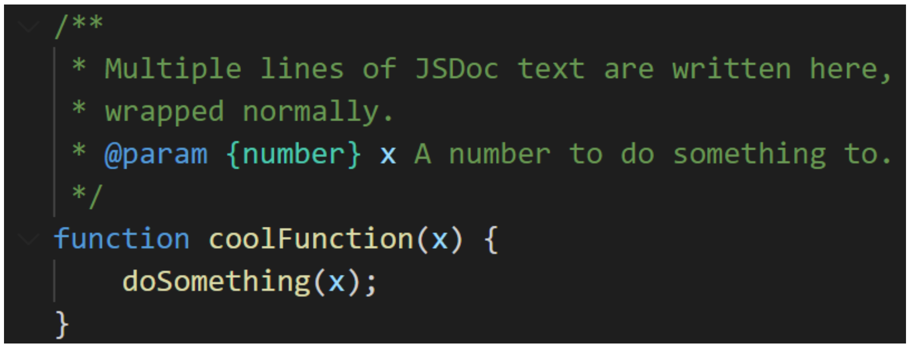
JSDoc:

General:

JSDoc should be used on all classes, fields, and methods.

General Form:

The basic formatting of JSDoc blocks can be seen in the example below.



```
/**
 * Multiple lines of JSDoc text are written here,
 * wrapped normally.
 * @param {number} x A number to do something to.
 */
function coolFunction(x) {
    doSomething(x);
}
```

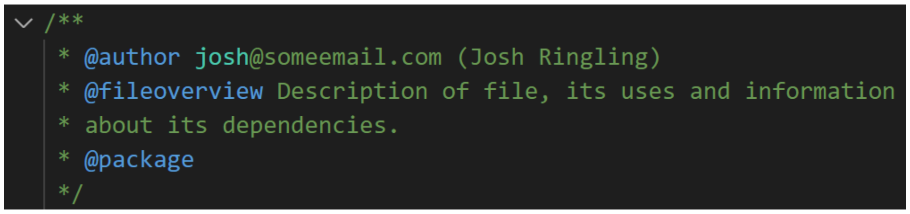
Line Wrapping:

JSDoc lines that go past the 80 character should be wrapped onto the next line and indented like any other code.

File Overview:

A file should have a top-level file overview which should include a description of the file, but can also include a copyright notice, author information, or any other relevant document related information.

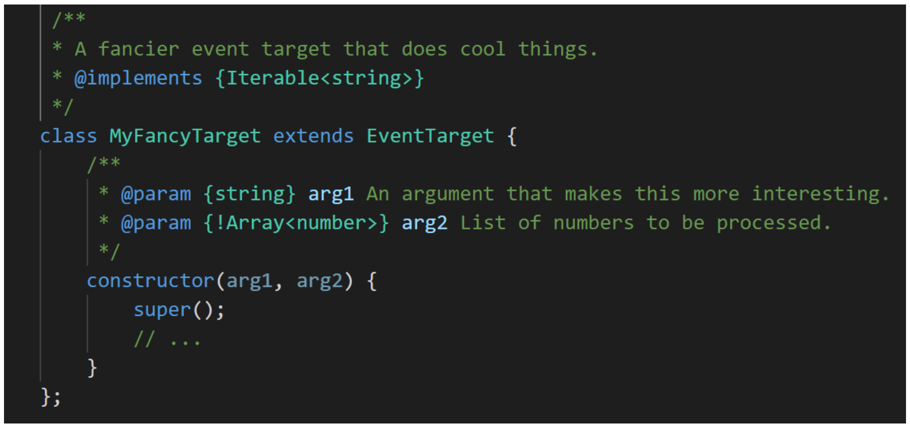
It is also recommended to include authors of files in the style of the below example.



```
/**
 * @author josh@someemail.com (Josh Ringling)
 * @fileoverview Description of file, its uses and information
 * about its dependencies.
 * @package
 */
```

Class Comments:

Classes and interfaces must be documented with a description and any template parameters and implemented interfaces. The class description should be detailed enough that the reader knows where this class could be used.



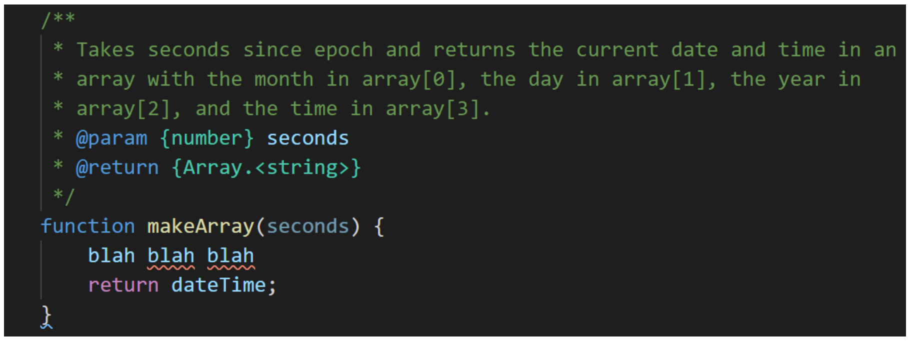
```
/**
 * A fancier event target that does cool things.
 * @implements {Iterable<string>}
 */
class MyFancyTarget extends EventTarget {
    /**
     * @param {string} arg1 An argument that makes this more interesting.
     * @param {!Array<number>} arg2 List of numbers to be processed.
     */
    constructor(arg1, arg2) {
        super();
        // ...
    }
};
```

Method and Function Comments:

In methods and named functions, parameter and return types must be documented.

Method descriptions begin with a verb phrase that describes what the method does.

If a method overrides a superclass method, it must include an @override annotation.



```
/**
 * Takes seconds since epoch and returns the current date and time in an
 * array with the month in array[0], the day in array[1], the year in
 * array[2], and the time in array[3].
 * @param {number} seconds
 * @return {Array.<string>}
 */
function makeArray(seconds) {
    blah blah blah
    return dateTime;
}
```

React Style Guide

Joshua Ringling: ring7885@stthomas.edu

Background:

This document defines the formatting and style rules for React. It applies to raw, working files that use React coding.

General Rules:

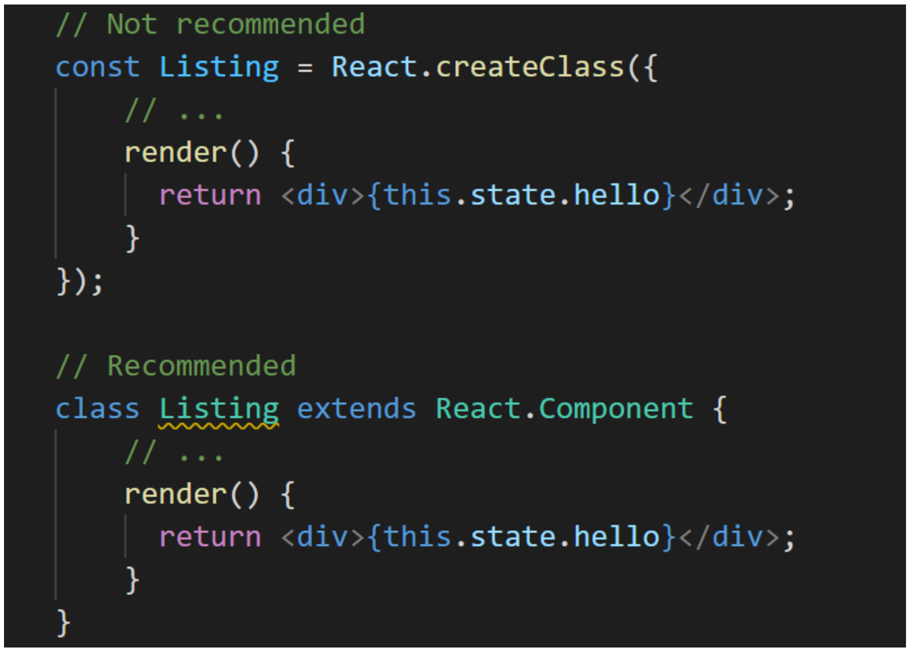
Basics:

Only include one React component per file.

Always use JSX syntax. While using JSX syntax you are expected to follow the style guide instructions for HTML.

Class vs. React.createClass:

If you have any references to internal state, extending React.Component is a lot cleaner.



```
// Not recommended
const Listing = React.createClass({
    // ...
    render() {
      return <div>{this.state.hello}</div>;
    }
});

// Recommended
class Listing extends React.Component {
    // ...
    render() {
      return <div>{this.state.hello}</div>;
    }
}
```

Mixins:

Do not use mixins. Just don’t use them.

Mixins introduce implicit dependencies and can cause clashing naming schemes.

Naming:

Extensions: Always use the .jsx extension for React components.

Filename: Use PascalCase for filenames.

References: Use PascalCase for React components and camelCase for their instances.

PascalCase is just like camelCase, but the first word is capitalized.

Declaration:

Do not use *displayName* for naming components.

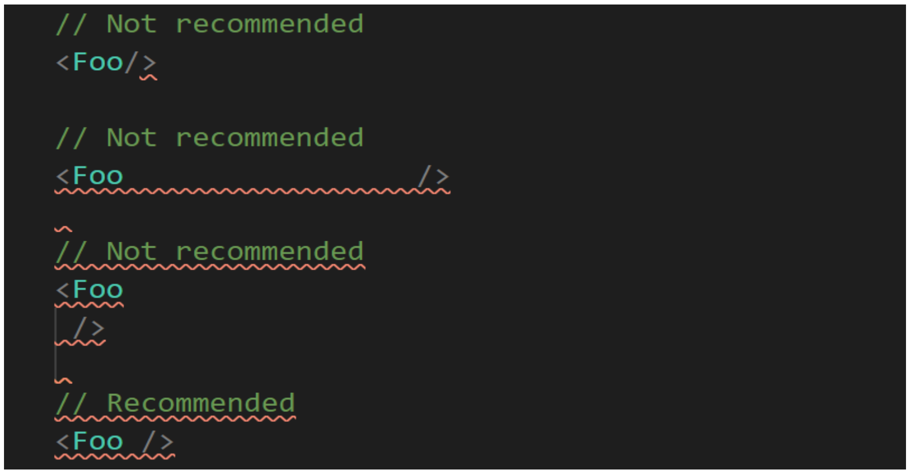
Quotes:

Always use double quotes (“) for JSX attributes.

Use single quotes (‘) for all other JavaScript uses.

Spacing:

Always include a single space in your self-closing tag.



```
// Not recommended
<Foo/>

// Not recommended
<Foo                 />

// Not recommended
<Foo
 />

// Recommended
<Foo />
```

Props:

Always use camelCase for prop names and PascalCase if the prop value is a React component.

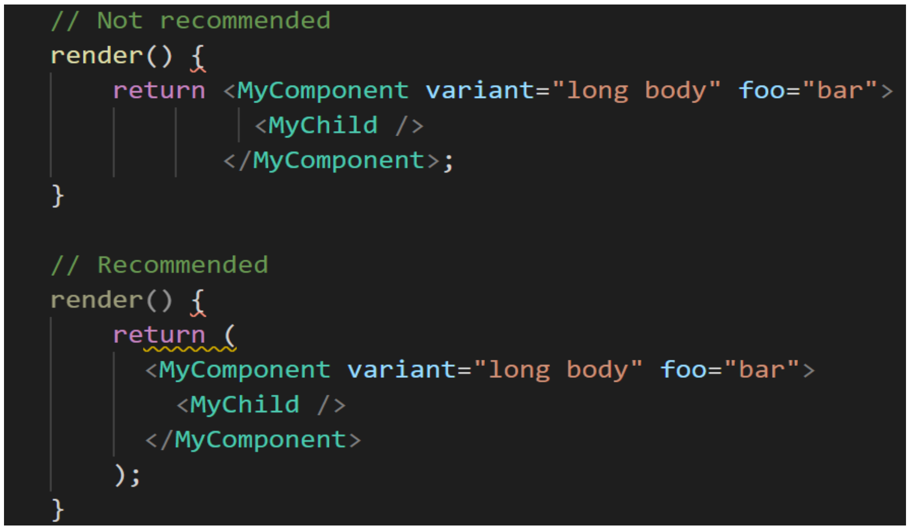
Always include an alt prop for <img> tags.

Do not use words like “image”, “photo”, or “picture” in <img> or alt props.

Do not use accessKey.

Parentheses:

If JSX tags span more than one line, wrap them in parentheses.

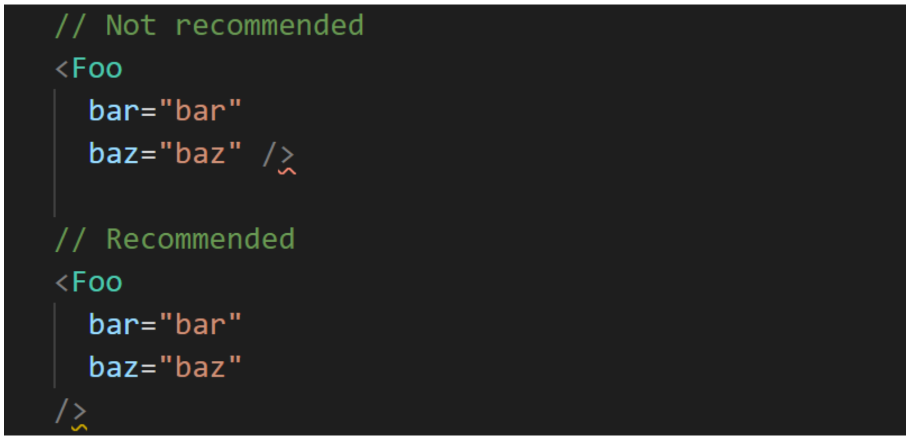


```
// Not recommended
render() {
    return <MyComponent variant="long body" foo="bar">
             <MyChild />
           </MyComponent>;
}

// Recommended
render() {
    return (
      <MyComponent variant="long body" foo="bar">
        <MyChild />
      </MyComponent>
    );
}
```

Tags:

Always self-close tags that have no body.

If your component has multiline properties, close its tag on a new line.



```
// Not recommended
<Foo
  bar="bar"
  baz="baz" />

// Recommended
<Foo
  bar="bar"
  baz="baz"
/>
```

Methods:

Arrow functions are allowed as long as they don't massively hinder performance like causing multiple pointless rerenders.

Bind event handlers for the render method in the constructor.

isMounted:

Do not use isMounted.

isMounted is not available when using ES6 classes and is soon to be officially deprecated.