# Preconditions

Use bad-code.js file that is attached to this homework as a starting point, it will contain simple codebase, and all needed parts to start.

# Task description

The goal of this task is to configure eslint and husky to get some experience with static code analysis.

\*Usage of eslint plugins are forbidden

# Evaluation criteria

For every day of lateness there is a penalty in 0.5 point.

Maximum 10 points.

1. The package contains husky hook on git-commit that runs linter [2 points].
2. ESlint is configured accordingly to all requirements and shows ~127 errors in bad-code.js [8 points].
   1. There is no possibility to declare variables with key word var
   2. Variables declared with const can’t be reassigned
   3. Usage of object method shorthand is forced

const atom = {

// bad

addValue: function (value) { … },

// good

addValue(value) { … },

};

* 1. Usage of property values shorthand is forced

const lukeSkywalker = 'Luke Skywalker';

// bad

const obj = {

lukeSkywalker: lukeSkywalker,

};

// good

const obj = {

lukeSkywalker,

};

* 1. Object properties should be quoted only if the property name can’t be used without quotes.

// bad

const bad = {

'foo': 3,

'bar': 4,

'data-blah': 5,

};

// good

const good = {

foo: 3,

bar: 4,

'data-blah': 5,

};

* 1. Single quotes are used for strings.
  2. Templates strings should be used instead of concatenation.

// bad

function sayHi(name) {

return 'How are you, ' + name + '?';

}

// good

function sayHi(name) {

return `How are you, ${name}?`;

}

* 1. Function declaration should have spaces between key word function call signature and body.

// bad

const f = function(){};

const g = function (){};

const h = function() {};

// good

const x = function () {};

const y = function a() {};

* 1. Spread operator should be used to call variadic functions.

// bad

new (Function.prototype.bind.apply(Date, [null, 2016, 8, 5]));

// good

new Date(...[2016, 8, 5]);

* 1. Functions with multiline signatures, or invocations, should be indented just like every other multiline list in this guide: with each item on a line by itself, with a trailing comma on the last item

// bad

function foo(bar,

baz,

quux) {

// ...

}

// good

function foo(

bar,

baz,

quux,

) {

// ...

}

// bad

console.log(foo,

bar,

baz);

// good

console.log(

foo,

bar,

baz,

);

* 1. Arrow functions should be always used when anonymous function is applied.

// bad

[1, 2, 3].map(function (x) { … });

// good

[1, 2, 3].map((x) => { … });

* 1. Arrow functions should always have parentheses around arguments

// bad

[1, 2, 3].map(x => { … });

// good

[1, 2, 3].map((x) => { … });

* 1. Line breaks should be avoided before or after = sign in an assignment statement

// bad

const foo =

superLongLongLongLongLongLongLongLongFunctionName();

// bad

const foo

= 'superLongLongLongLongLongLongLongLongString';

// good

const foo = (

superLongLongLongLongLongLongLongLongFunctionName()

);

// good

const foo = 'superLongLongLongLongLongLongLongLongString';

* 1. === and !=== should be used over == and != signs.

// bad

const foo = a ? a : b;

const bar = c ? true : false;

const baz = c ? false : true;

// good

const foo = a || b;

const bar = !!c;

const baz = !c;

* 1. Curly braces should be used with all multi-line blocks.

// bad

if (test)

return false;

// good

if (test) return false;

// good

if (test) {

return false;

}

// bad

function foo() { return false; }

// good

function bar() {

return false;

}

* 1. If you’re using multi-line blocks with if and lese, put else on the same line as your if block’s closing brace.

// bad

if (test) {

thing1();

thing2();

}

else {

thing3();

}

// good

if (test) {

thing1();

thing2();

} else {

thing3();

}

* 1. In case if block always executes a return statement, the subsequent else block is unnecessary.

// bad

function foo() {

if (x) {

return x;

} else {

return y;

}

}

// bad

function cats() {

if (x) {

return x;

} else if (y) {

return y;

}

}

// bad

function dogs() {

if (x) {

return x;

} else {

if (y) {

return y;

}

}

}

// good

function foo() {

if (x) {

return x;

}

return y;

}

// good

function cats() {

if (x) {

return x;

}

if (y) {

return y;

}

}

// good

function dogs(x) {

if (x) {

if (z) {

return y;

}

} else {

return z;

}

}

* 1. All comments should have a space at the beginning.
  2. Soft tabs (space character) should be used and set to 2 spaces.
  3. Leading curly brace should have a space before it.

// bad

function test(){

console.log('test');

}

// good

function test() {

console.log('test');

}

// bad

dog.set('attr',{

age: '1 year',

breed: 'Bernese Mountain Dog',

});

// good

dog.set('attr', {

age: '1 year',

breed: 'Bernese Mountain Dog',

});

* 1. Place 1 space before the opening parenthesis in control statements (if, while etc.). Place no space between the argument list and the function name in function calls and declarations

// bad

if(isJedi) {

fight ();

}

// good

if (isJedi) {

fight();

}

// bad

function fight () {

console.log ('Swooosh!');

}

// good

function fight() {

console.log('Swooosh!');

}

* 1. Operators should be surrounded by spaces

// bad

const x=y+5;

// good

const x = y + 5;

* 1. When making long method chains (more than 2 method chains) every method call should be moved onto new line using leading dot, which emphasizes that the line is a method call.

// bad

const leds = stage.selectAll('.led').data(data).enter().append('svg:svg').classed('led', true)

.attr('width', (radius + margin) \* 2).append('svg:g')

.attr('transform', `translate(${radius + margin},${radius + margin})`)

.call(tron.led);

// good

const leds = stage.selectAll('.led')

.data(data)

.enter().append('svg:svg')

.classed('led', true)

.attr('width', (radius + margin) \* 2)

.append('svg:g')

.attr('transform', `translate(${radius + margin},${radius + margin})`)

.call(tron.led);

* 1. Code blocks should not contains paddings with blank lines.

// bad

function bar() {

console.log(foo);

}

* 1. Parentheses and square brackets should not contains spaces.

// bad

function bar( foo ) {

return foo;

}

// bad

const foo = [ 1, 2, 3 ];

// good

console.log(foo[0]);

* 1. Curly braces should have spaces inside a block

// good

const foo = { clark: 'kent' };

// bad

const foo = {clark: 'kent'};

* 1. You should use only leading commas

// bad

const story = [

once

, upon

, aTime

];

// good

const story = [

once,

upon,

aTime,

];

* 1. In case an argument is placed on a new line it should contain trailing comma.

// bad

function createHero(

firstName,

lastName,

inventorOf

) {

// does nothing

}

// good

function createHero(

firstName,

lastName,

inventorOf,

) {

// does nothing

}

* 1. Semicolon should be used always

// bad

const luke = {}

const leia = {}

[luke, leia].forEach(jedi => jedi.father = 'vader')

// good

const luke = {};

const leia = {};

[luke, leia].forEach((jedi) => {

jedi.father = 'vader';

});