Specification of Source Style Guide—Version 1.0 edition

Martin Henz

National University of Singapore
School of Computing

March 5, 2020

This is the style guide for the language Source, the official language of the book *Structure and Interpretation of Computer Programs*, JavaScript Adaptation. You have never heard of Source? No worries! It was invented it just for the purpose of the book. Source is a sublanguage of ECMAScript 2016 (7th Edition). This style guide is a compilation of commonly accepted rules, with the possible exception of conditional expressions on page **??**.

Indentation

Use a four space indent. This means that if a line L ends with `{`, then the following line starts with four more spaces than required to reach the keyword **`function`**, **`if`** or **`else`** in L. Example:

```
function make_abs_adder(x) {
    function the_adder(y) {
        if (x >= 0) {
            return x + y;
        } else {
            return -x + y;
        }
    }
    return the_adder;
}
```

Note the four spaces before **`return`**, and then indentation of the **`if`** statement four spaces to the right of the preceding **`function`**.
Four space indent also applies when individual lines get too long and need to be broken at convenient places. Example:

```
function frac_sum(a, b) {
    return (a > b) ? 0
        : 1 / (a * (a + 2))
            +
            frac_sum(a + 4, b);
}
```

Line Length

Lines should be truncated such that they do not require excessive horizontal scrolling. As a guide, it should be *no more than 80 characters*.

Curly Braces

Curly braces should *open on the same line and close on a new line.* Statements within the curly braces should be indented one more level.

```
// function declaration
function my_function(<parameters>) {
    <statements here should be indented>
}

// if-else
if (<predicate>) {
    ...
} else if (<predicate>) {
    ...
} else {
    ...
}

// nested-if
if (<predicate>) {
    ...
    if (<some other predicate>) {
        ...
    }
    ...
}
```

Always use curly braces. Even if the block consists of only one statement. This is required in Source, and recommended in JavaScript.

```
// correct Source
if (<predicate>) {
    return x + y;
} else {
    return x * y;
}

// incorrect Source
if (<predicate>)
    return x + y;
else
    return x * y;

// worse
if (<predicate>) return x + y;
else return x * y;
```

Whitespace

Operators

Leave a single space between binary and ternary operators.

```
// good style
const x = 1 + 1;

// bad style
const x=1+1;

// good style
return x === 0 ? "zero" : "not zero";

// bad style
return x === 0?"zero":"not zero";
```

Do not leave a space between unary operators and the variable involved.

```
// good style
const negative_x = -x;

// bad style
const negative_x = - x;
```

Function Definition Expressions

Keep the parameters and the body expression of a function definition expression in one line, if they are short enough.
If they are lengthy, use indentation. The indentation starts four characters after first character of the *parameter*, if there is only one, or four characters after the open parenthesis, if there are multiple parameters. If the body expression does not fit in one line, use indentation following the first line of the body expression.

```
// good style
function count_buttons(garment) {
    return accumulate((sleaves, total) => sleaves + total,
        0,
        map(jacket =>
            is_checkered(jacket)
                ? count_buttons(jacket)
                : 1,
                garment));
}

// good style
function count_buttons(garment) {
    return accumulate(
        (sleaves, total) =>
            delicate_calculation(sleaves + total),
        0,
        map(jacket =>
            is_checkered(jacket)
                ? count_buttons(jacket)
                : 1,
            garment));
}

// bad style: too much indentation
function count_buttons(garment) {
    return accumulate((sleaves, total) =>
                      delicate_calculation(sleaves + total),
                      0,
                      map(jacket =>
                          is_checkered(jacket)
                          ? count_buttons(jacket)
                          : 1,
                          garment));
}

// no newline allowed between parameters and =>
function count_buttons(garment) {
    return accumulate(
        (sleaves, total)
            => delicate_calculation(sleaves + total),
        0,
        map(jacket
            => is_checkered(jacket)
                ? count_buttons(jacket)
                : 1,
            garment));
}
```

Conditional Expressions

Keep the three components of a conditional expression in one line, if they are short enough.

```
// good style
const aspect_ratio = landscape ? 4 / 3 : 3 / 4;

// bad style
const aspect_ratio = landscape
    ? 4 / 3
    : 3 / 4;
```

If the *consequent-expression* or *alternative-expression* are lengthy, use indentation. The indentation is as usual four characters longer than the indentation of the previous line.

```
// good style
function A(x,y) {
    return y === 0
        ? 0
        : x === 0
            ? 2 * y
            : y === 1
                ? 2
                : A(x - 1, A(x, y - 1));

// bad style: line too long
function A(x,y) {
    return y === 0 ? 0 : x === 0 ? 2 * y : y === 1 ? 2 : A(x - 1, A(x, y - 1));
}

// bad style: too much indentation
function A(x,y) {
    return y === 0
               ? 0
               : x === 0
                     ? 2 * y
                     : y === 1
                           ? 2
                           : A(x - 1, A(x, y - 1));
}
```

Conditional Statements and Functions

Leave a single space between the **if** statement and the first parenthesis and before every opening curly brace. Start your **else** statement on the same line as the closing curly brace, with a single space between them.

```
if (<predicate>) {
    ...
} else if (<predicate>) {
    ...
} else {
    ...
}
```

When calling or declaring a function with multiple parameters, leave a space after each comma. There should also be no spaces before your parameter list.

```
// good style
function my_function(arg1, arg2, arg3) {
    ...
}

// bad style
function my_function (arg1, arg2, arg3) {
    ...
}

// good style
my_function(1, 2, 3);

// bad style
my_function(1,2,3);

// bad style
my_function (1, 2, 3);
```

There should be no spaces after your opening parenthesis and before your closing parenthesis.

```
// bad style
function my_function( arg1, arg2, arg3 ) {
    ...
}

// bad style
my_function( 1, 2, 3 );

// bad style
if ( x === 1 ) {
    ...
}

// good style
function my_function(arg1, arg2, arg3) {
    ...
}

// good style
my_function(1, 2, 3);

// good style
if (x === 1) {
    ...
}
```

Clean up *all trailing whitespace* before submitting your program.

Names

Choice of names

When naming constants or variables, use *underscores* to separate words. Examples: `my_variable`, `x`, `rcross_bb`.

Nesting

Do not use the same name for nested scope. Examples:

```
// bad program
```

```
const x = 1;
function f(x) {
    // here, the name x declared using const
    // is ''shadowed'' by the formal parameter x
    ...
}

// another bad program
function f(x) {
    return x => ...;
    // here, the formal parameter x of f is ''shadowed''
    // by the formal parameter of the returned function
}

// a third bad program
function f(x) {
    const x = 1;
    // in the following, the formal parameter x of f
    // is ''shadowed'' by the const declaration of x.
    ...
}
```

Finally, the worst case would be a (surely accidental) use of the same variable name for two parameters of a function. In this case, the second variable is not visible; it is “shadowed” by the first.

```
// worse than the above
function f(x, x) {
    ...
}
```

Comments

Comments should be used to describe and explain statements that might not be obvious to a reader. Redundant comments should be avoided. The comment in the following program is useful because it explains what `x` and `y` stands for and what type of object is meant.

```
// area of rectangle with sides x and y
function area(x, y) {
    return x * y;
}
```

The programmer has decided to use the short word `area` as the name of the function, which is ok, as long as it is clear that the geometric objects that the program deals with are always rectangles.
An example for bad style as a result of a redundant comment follows here:

```
// square computes the square of the argument x
function square(x) {
    return x * x;
}
```

For multi-line comments, use `/* ... */` and for single line comments, use `//`.