National University of Singapore
School of Computing
CS1101S: Programming Methodology
Semester I, 2017/18

Source Style Guide

Introduction

The language Source is the official language of CS1101S. You have never heard of Source? No wonder, because we invented it just for the purpose of this module. It is a strict sublanguage of JavaScript Version 1.8.5/ECMAScript 5 and as such is fully supported by our development environment (Firefox, Chrome, as well as The Source Academy).

This document contains guidelines regarding the coding style to be used when writing Source. Students will receive deductions of marks starting from Week 4 if they violate these guidelines.

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

Note the four spaces before **`return`**, and then indentation of the **`if`** statement four spaces to the right of the preceding **`function`**.

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
var x = 1 + 1;

// bad style
var 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
var negative_x = -x;

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

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.

Variables

Naming

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

Nesting

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

```
// bad program
var x = 1;
function f(x) {
    // here, the variable x declared using var
```

```
    // is ``shadowed'' by the formal parameter x
    ...
}

// another bad program
function f(x) {
    return function(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) {
    var x = 1;
    // in the following, the formal parameter x of f
    // is ``shadowed'' by the var 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 `//`.