Coding Conventions

or...what your code should look like

Employing good and consistent coding conventions is very important. It helps make the programs readable and improves productivity (not to mention the fact that it is very helpful for your TAs). For these reasons, it is in your best interest that all assignments meet the standards described below.

Don't worry if you are not sure what the terms below mean; you will soon get to know them thoroughly. After reading this document, make sure you keep it by your side when you code to answer your questions about format and style.

Names

Using the following naming conventions will keep code ambiguities to a minimum.

Procedure Names Example: `(square)`

These usually perform some kind of action and/or return a value. We use nouns to describe the returned value.

Predicates Examples: (empty?)

Predicates that return boolean values end in a question mark "?". We use verbs (is-odd?) or adjectives (odd?) to name such predicates.

Note:

You should generally avoid using one-letter variable names, as they are not descriptive and will make your code more difficult to read. The only exceptions are if the usage agrees with an established naming convention,

such as `x` and `y` for coordinates. Always choose the most descriptive name possible.

Indentation and Skipping Lines

Scheme consists of nested blocks of code. For example, an `if` statement can be nested within another `if` statement, which in turn is contained in a procedure. One way to show this nesting is by indentation. This allows the person reading your code to easily see the major sections of code and main flow-of-control in methods.

Here are indentation conventions for CS3:

Nested code should be indented.

```
(define (sign number)
    (if (< number 0)
         'negative
         (if (= number 0)
             'zero
             'positive)))
```

Skipping Lines

Another way of making major sections of your code easy to see is by skipping lines. You should skip about two lines between procedures. You should also separate logical chunks of code within a procedure with one or two blank lines.

Internal Documentation (a.k.a. Comments)

"Internal documentation" simply refers to the comments in your program. No matter how simple a program is to its author, it is almost always confusing to another person reading it (i.e, the TAs who will be grading your programs). Comment well, and you will have a happy TA. Comment poorly, and the TA will

have trouble understanding what you are doing (and you may not get the benefit of the doubt).

Comments in Scheme are indicated by a semicolon. Scheme ignores everything to the right of a semicolon.

Example:

```
; This is a one line comment
```

Comments can be added to the right of a line of code, to explain what that line is doing.

Example:

```
 (if (equal? x 'monday)    ;Check to see if x is monday
```

```
;; Comments that are several lines long
;; are indicated by double semicolons.
```

Comments that are several lines long are included right before procedures.

Comments are used for adding information to your code. **Your comments should not just repeat the code that you have written, as such comments are useless. Comments should be added whenever an explanation or clarification is needed for some section or line of code,** such as a difficult sequence of commands, an unusual or atypical design pattern, or a very brief synopsis of what a particular method call might do or return.

Comment as you go along. DO NOT leave all your commenting until the end of the project. Commenting done later takes more time because you have to remember or figure out what the code is doing instead of just writing down what you are already thinking about.

Header Comments

Header comments describe procedures. They consist of a block of information placed before the code they explain. Use multiline

commenting style (`;;`) for header comments. You should describe each procedure and its design in the header comment. Since you use one file for each program, remember to put your ***login name*** and your ***name*** at the top of each file.

Every procedure should have a header which contains a comment describing its purpose, how it achieves its purpose, and a list of placeholders used as arguments to the method. Each placeholder should be described also. The return value for the procedure should be described in the header as well.

Inline Comments

Inline comments appear in code clarifying its workings.

Lines should be commented to describe what they are doing. Note that this comment should not just repeat what the Scheme code does, but should elaborate on the code. This should be done for every section and logical chunk of code.

```
_(+ days 1)  ;add 1 to x                          <- useless
_(+ days 1)  ;adjust days for Leap Year           <- useful
```