CSE 341, Fall 2004, Assignment 4 (version 1)
Due: Wednesday 10 November, 9:00AM

Set-up: This assignment requires library code and some code your instructor wrote. Do *all* the following:

1. Set the DrScheme language level to "Pretty Big (includes MrEd and Advanced)" in the "PLT" list.
2. Add the `draw.ss` Teachpack available in the `teachpack/htdp` subdirectory of `PLT`.
3. Download `hw4support.scm` from the course website and put it in the same directory as your solution.
4. Begin your solution with `(load "hw4support.scm")`.

Provided code: The following will help you understand the provided code, but you don't need to understand the details (just how to use `display-piece`, `display-stream`, and `sleep-for-a-while`):

- Read the documentation for `draw.ss` (Go to Help, then Help Desk, then Teachpacks, then Simple Drawing Exercises. Ignore the first four functions; they are not used.)
- Then read through `hw4support.scm` with the remaining ideas in mind.
- The 7 Tetris pieces are represented just like in homework 1. You don't need `all-pieces` except for the extra credit.
- `draw-graphics-square`, `convert-coord`, and `draw-tetris-piece` are just helper functions, but `draw-tetris-piece` *will not work* until you do problem 2.
- `display-tetris-piece` takes a list of pairs (e.g., `tetris-L1`) and displays the corresponding piece in a window. Mouse-clicking in the window makes it disappear. Your pair coordinates should be between `-4` and `4` (inclusive) or you won't see them.
- `display-stream` takes a number n and a piece-stream, where a piece-stream is a thunk that returns a piece and a piece-stream. `display-stream` presents a blank window. Clicking in it shows the first piece in the piece-stream, clicking again shows the second piece, and so on. After $n + 1$ clicks, the window disappears.

If you close the display-window manually (i.e., not by clicking in it the right number of times), then you'll have to hit the Stop button in DrScheme.

Mutation rules: You must not use anything with a `!` character (`set!`, `set-car!`, or `set-cdr!`) except in problem 7, where you need to. (You will also want to use mutation to *test* problem 8, but not to solve it.)

Warning: Only the first two problems are "warm-up" exercises for Scheme. After that we dive right into streams, memo-tables, and macros, which are nontrivial concepts.

1. Write a function `turn-clockwise` that takes a Tetris piece p and a number n and returns a piece that is like p rotated 90-degrees clockwise n times. It is unnecessary to "spin the piece in place" so here's a simple and correct approach: If p has a square (x, y), then turning p 90-degrees clockwise puts a square at $(y, -x)$. Do *not* assume pieces have 4 squares. Sample solution: 5 lines (using the `map` primitive).
2. Write a function `list-iter` that takes a function f and a list *lst*, applies f to each element of *lst* in order, *ignores the results*, and returns `#t`. Sample solution: 5 lines. As explained above, you can use `display-tetris-piece` now, which should make it easy to test `turn-clockwise` and `list-iter`.
3. Write a piece-stream `all-ts` where every piece in the stream is the piece bound to `tetris-T`. That is, `all-ts` is a thunk that produces a pair where the first component is the "T" piece and the second component is a piece-stream conataining all "T" pieces. Sample solution: 1 line.
4. Write a piece-stream `alternate-Ls` where the pieces in the stream alternate between `tetris-L1` and `tetris-L2` (start with `tetris-L1`). Hint: Use mutually recursive functions. Sample solution: 4 lines.

5. Write a piece-stream `four-turns-L1` where the pieces in the stream cycle through 4 distinct pieces: `tetris-L1` rotated clockwise 0, 90, 180, and 270 degrees. Hint: Use mutually recursive functions. Sample solution: 10 lines. (Shorter solutions are possible; the sample avoids repeating computations.)

6. Write a function `alternate-streams` that takes two piece-streams (call them s_1 and s_2) and returns a stream (call it s_3) that alternates between pieces from s_1 and s_2: The pieces in s_3 should be "first from s_1", "first from s_2", "second from s_1", etc. Hint: Have `alternate-streams` return a thunk that when invoked calls `alternate-streams` (in addition to other things). Sample solution: 4 (rather clever) lines.

7. Write a function `bad-memory-penalizer` that (unlike memoization) makes a function that is *slower* when called with the same arguments. `bad-memory-penalizer` takes a function `f` and returns a function (call it g) equivalent to `f`. However, when g is called with the "same" value v for the n^{th} time (for $n \geq 1$), it must first call `sleep-for-a-while` with n and *then* call `f` with v and return the result. The definition of "same" is the definition used by the primitives `equal?` and `assoc` (the sample solution just uses `assoc`). Hint: You will need a mutable table to remember how many times each value has been passed to g.

8. Write a macro `whileNotX` such that `(whileNotX` e_1 e_2 e_3`)` does the following:

   - It evaluates e_2 to a value that is presumably a number x.
   - While `(=` e_1 x`)` is `#f`, it evaluates e_3.
   - Once `(=` e_1 x`)` is `#t`, the whole `whileNotX` expression is also `#t`.

   Note that e_2 is evaluated exactly once whereas e_1 and e_3 may be evaluated in any number of times. Hint: Define a recursive thunk. Example:

   ```
   (define a 7)
   (define b 4)
   (whileNotX a b (begin (set! b (+ b 2))(set! a (- a 1))))
   ```

   will make `a` bound to `4` and `b` bound to 10. If you then evaluate
   `(whileNotX a b (begin (set!  b (+ b 2))(set!  a (- a 1))))`
   again, you will enter an infinite loop.

Extra Credit:

EC1 Write a piece-stream `alternate-all` that cycles through the pieces in the Scheme vector `all-pieces`. Do *not* assume `all-pieces` has length 7. Hint: Use `remainder`.

EC2 Write a function `n-turns` that takes a number n (greater than 0) and a piece-stream s_1 and returns a piece-stream (call is s_2). The first n pieces in s_2 should be the *first* piece in s_1, but rotated 0, 90, 180, ... degrees clockwise. The next n pieces in s_2 should be the *second* piece in s_1, but rotated 0, 90, 180, ... degrees clockwise. And so on. For example, `(n-turns 4 alternate-all)` should cycle through 28 pieces (although some may look the same since, for example, turning a square still makes a square).

Turn-in Instructions

- Put all your solutions in one file, `lastname_hw4.scm`, where `lastname` is replaced with your last name.
- The first line of your `.scm` file should be a Scheme comment with your name and the phrase `homework 4`.
- Email your solution to `brianhk@cs.washington.edu`.
- The subject of your email should be *exactly* `[cse341-hw4]`.
- Your `.scm` file should be an *attachment*.