CSE 341, Winter 2009, Assignment 3
Scheme Warmup
Due: Wed Jan 28, 10:00pm

18 points total (2 points each for Questions 1–4; 10 points for Question 5)

You can use up to 3 late days for this assignment.

1. Write a function `take` that takes an integer `n` and a list `s`, and returns a new list consisting of the first `n` elements in `s`. Handle the cases of the length of `s` being less than `n`, and `n` being 0 or negative, in the same way that Haskell does for its definition of `take`.

2. Write a *recursive* function `double-plus` that takes a list of integers, and returns a new list of integers of the same length, with each element in the new list computed using the formula $2n+1$. For example, `double-plus [1,3,10]` should evaluate to `[3,7,21]`, while `double-plus []` should evaluate to `[]`.

3. Write another version of the `double-plus` function, called `map-double-plus`, that uses the built-in `map` function in Scheme. `map-double-plus` should not be recursive. Don't define a named helper function to compute $2n+1$ — use an anonymous function.

4. Write a tail-recursive function `same-length` that takes two lists and returns true if they have the same length, and otherwise false. (You can assume the arguments are both lists.) Don't use the built-in `length` function.

5. This question is based on the symbolic differentiation program linked from the 341 Scheme web page. That program adapts an example in Chapter 2 of the book *Structure and Interpretation of Computer Programs*. The full text is available online (linked from the Scheme page); there is also a copy on reserve in the Engineering Library. The symbolic differentiation program linked from the 341 web page is somewhat modified however: it doesn't use constructor functions, and it separates out finding the derivative from simplifying expressions.

   First, load the symbolic differentiation program into Scheme and try it, to make sure it is working OK and that you understand it. Now add the following extensions, by allowing the expressions to be differentiated to include:

   - the difference operator
   - sin and cos
   - raising an expression to an integer power

   Difference should be really easy — use sum as a model. The rules for the others are as follows:

$$\frac{d(\sin u)}{dx} = \cos u(\frac{du}{dx})$$

$$\frac{d(\cos u)}{dx} = -\sin u(\frac{du}{dx})$$

$$\frac{d(u^n)}{dx} = nu^{n-1}(\frac{du}{dx})$$

   For sin and cos, just simplify applying these to a number, e.g. $\sin 0$ should simplify to 0. For simplification of the power function build in the rules that anything to the 0 power is 1, and anything to the power 1 is itself. Also simplify a number raised to another number, e.g. 3^2 should simplify to 9. Your code for this question should be written entirely in a functional style — no side effects.

Turnin: Your program should include some well-chosen unit tests for each of your functions. As usual, your program should be tastefully commented (i.e. put in a comment before each function definition saying what it does), and in good style.