CSE 341, Winter 2010, Assignment 3
Scheme Warmup
Due: Friday Jan 29, 10:00pm

20 points total (2 points each for Questions 1–5; 10 points for Question 6). Some of these questions are reprises of questions in the Haskell warmup (the idea being that you end up comparing the languages). It's fine to look at the sample solution to the Haskell warmup or your own Haskell code in writing the Scheme questions.

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

1. Write a function `sphere_area` that takes a number representing the radius of a sphere and returns the surface area of that sphere.
2. 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`.
3. Write a *recursive* function `squares` that takes a list of numbers, and returns a new list of the squares of those numbers.
4. Write another version of the `squares` function, called `map-squares`, that uses the built-in `map` function in Scheme. `map-squares` itself should not be recursive. Don't define a named helper function to compute each square — use an anonymous function.
5. Write a `ascending` function to test whether a list of integers is in strict ascending order. For example, `ascending [1,2,5]` should return true, while `ascending [2,3,1]` and `ascending [2,2]` should both return false. You should handle the empty list, and a list of one number. (What should these return? Justify your decision in a comment in the code.) Also say in a comment whether or not your `ascending` function is tail-recursive.
6. 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.