Assigned: Tuesday 3 September 2013
Due: Monday 9 September 2013 11:59 pm

Objectives:

- Reinforce Scheme basics
- Recall how to write recursive procedures over lists and numbers using named let and tail recursion
- Explore clever uses of higher-order procedures

Collaboration: This homework assignment must be completed individually.

A common problem in the AI subfield of machine learning is to do regression on a set of points. That is, assume points are observations from some unknown function, and we wish to infer the function so that we may make predictions by interpolating between these points.\(^1\) One approach to this problem is to model the unknown function as a polynomial, which means one must determine the coefficients. In this assignment, we’ll develop some useful Scheme functions for defining and evaluating “encapsulated” polynomials.

1. Add the procedures left-section, right-section, and dot-product from your in-class lab. Be sure to cite the originals appropriately and attribute the contributions of your partner(s).

2. Using named let\(^2\) and tail recursion\(^3\), write the procedure (iota num) that produces the list of numbers from 0 to num – 1 in order. For example,

   > (iota 5)
   (0 1 2 3 4)

   Your solution should not call reverse. If your original attempt produces results in the wrong order, you must think about building the result in the opposite order.

3. Write a procedure, (polynomial-term c n) that returns the function \(f(x) = c \cdot x^n\).

   > (define two-x-cubed (polynomial-term 2 3))
   > (two-x-cubed 1) ; 2 * 1 * 1 * 1
   2
   > (two-x-cubed 3) ; 2 * 3 * 3 * 3
   54
   > (define three-x-squared (polynomial-term 3 2))
   > (three-x-squared 1) ; 3 * 1 * 1
   3
   > (three-x-squared 5) ; 3 * 5 * 5
   75
   > ((polynomial-term 5 4) 2) ; 5 * 2^4
   80

4. Write a procedure (polynomial coeffs) that takes a list of coefficients for the terms \(x^0, x^1, x^2, \ldots\) of a polynomial and produces a function that takes a single value, evaluating the polynomial given those coefficients at that value.

\(^1\)For example, see AIMA section 18.2, pp. 695–697.
\(^2\)If you’ve forgotten how to write a named let, see, e.g., Dybvig, *Recursion and Iteration* http://www.scheme.com/tspl4/control.html#./control:s20
\(^3\)If you’ve forgotten about tail recursion, see e.g., Davis et al. *Helper Recursion*, http://www.cs.grinnell.edu/~weinman/courses/CSC261/2013F/misc/recursion-patterns.html
Create the polynomial \( f(x) = 1 \cdot x^0 + 4 \cdot x^1 = 1 + 4x \)

\[
(\text{define line (polynomial (list 1 4))})
\]

\[
(\text{line 5) ; Evaluate } f(5) = 1 + 4 \cdot 5
gen(21)
\]

Create the polynomial \( g(x) = 1 + 4x + 3x^2 - 2x^3 \)

\[
(\text{define cubic (polynomial (list 1 4 3 -2))})
\]

\[
(\text{cubic 5) ; Evaluate } g(5) = 1 + 4 \cdot 5 + 3 \cdot 5^2 - 2 \cdot 5^3
gen(-154)
\]

**Note:** Make your solution as concise as possible and do not (explicitly) use recursion.

5. Recall that the derivative of a polynomial-term may be given by \( \frac{d}{dx} c \cdot x^n = c \cdot n \cdot x^{n-1} \). Write a procedure (polynomial-derivative-coefs coeffs) that takes a list of coefficients and produces a list of coefficients of polynomial's derivative.

\[
(\text{polynomial-derivative-coefs (list 1 4 3 -2))}
\]

\[
(4 6 -6)
\]

6. It can often be tedious to manually compose the same function with itself several times. Often, we do not know in advance how many times a procedure should be applied.

Write a new procedure, nest, as follows. nest takes two parameters, a unary procedure \( f \) and an integer \( n \). The value produced is a new procedure that results from composing together \( n \) copies of \( f \).

Note that \( n \) must be at least 1 for nest to make sense.

\[
(\text{define plus5 (nest (+ 1) 5))}
\]

\[
(\text{plus5 6)}
gen(11)
\]

\[
(\text{define duplicate (lambda (val n) ((nest (left-section cons val) n) null))})
\]

\[
(\text{duplicate "hello" 5)}
\]

\[
("hello" "hello" "hello" "hello" "hello")
\]

\[
(\text{define second-derivative (nest polynomial-derivative-coefs 2))}
\]

\[
(\text{second-derivative (list 1 4 3 -2))}
\]

\[
(6 -12)
\]

7. Write a procedure (polynomial-deriv coeffs n) that takes a list of polynomial coefficients coeffs and a strictly positive integer \( n \) and produces a procedure that takes a single value and evaluates the \( n \)th derivative of the polynomial with the given coefficients at that value.

\[
(\text{define d2/dx2-cubic (polynomial-deriv (list 1 4 3 -2) 2))}
\]

\[
(\text{d2/dx2-cubic 5)}
\]

\[
(-54)
\]

8. Write a procedure (non-zero-coefficients coeffs) that takes a list of coefficients for the terms \( x^0, x^1, x^2, \ldots \) of a polynomial with terms \( c_n \cdot x^n \) and produces the values of \( n \) (in ascending order) for which \( c_n \neq 0 \).

\[
(\text{define (non-zero-coefficients (list 1 4 0 -2))})
\]

\[
(0 1 3)
\]

\[
(\text{define (non-zero-coefficients (polynomial-derivative-coefs (list 1 4 0 -2)))})
\]

\[
(0 2)
\]
What to turn in

Your submission should include the following

- Your completed implementation of all methods above in a .scm file
- A short driver program (.scm file) that demonstrates your procedures are correct (use examples other than those provided)
- A single PDF containing (merged)
  - Your Scheme files
  - A transcript of your driver program’s output

Files in any other format will receive a zero.

Acknowledgements

Problems 3 and 4 are adapted from S. Rebelsky and J. Weinman, Exam 3: Sophisticated Scheming, CSC151 2010S. Problem 6 is adapted from J. Davis and J. Weinman, Exam 3: Sophisticated Scheming, CSC151 2011S.