1. Type each of the following into Scheme, and note the results. See when you can predict the results before letting Scheme do the computation.

(lambda (x) (+ x 3))

((lambda (x) (+ x 3)) 7)

You can think of lambda as meaning “the function of...,” e.g., “the function of x that returns (+ x 3).

(define (make-adder num)
  (lambda (x) (+ x num)))

((make-adder 3) (make-adder 3) (make-adder 3) 7)

(define plus3 (make-adder 3))

(plus3 7)

(define (square x) (* x x))

(square 5)

(define sq (lambda (x) (* x x)))

(sq 5)

(define (try f) (f 3 5))

(try +)

(try word)

2. Suppose we have the following definitions:

(define (foo num) (* num 2))

(define (bar fn) (lambda (num) (fn (+ num 2))))

(define (baz fn) (lambda (num) (+ (fn num) 2)))

- What does foo take as input? What does foo output?
- What does bar take as input? What does bar output?
What type of data will calling bar on foo, i.e. (bar foo) return?

• How many procedure calls do we make TOTAL if we evaluate ((bar foo) 2)?

• Will ((bar foo) 3) and ((baz foo) 3) give the same answer? Think about it yourself a bit before consulting STk.

3. Write a procedure substitute that takes three arguments: a sentence, an old word, and a new word. It should return a copy of the sentence, but with every occurrence of the old word replaced by the new word. For example:

> (substitute '(she loves you yeah yeah yeah) 'yeah 'maybe)
(she loves you maybe maybe maybe)

4. Consider a Scheme function g for which the expression

((g) 1)

returns the value 3 when evaluated. Determine how many arguments g has. In one word, also describe as best you can the type of value returned by g.

5. For each of the following expressions, what must f be in order for the evaluation of the expression to succeed, without causing an error? For each expression, give a definition of f such that evaluating the expression will not cause an error, and say what the expression’s value will be, given your definition.

f
(f)
(f 3)
((f))
(((f)) 3)

6. Find the values of the expressions

((t 1+) 0)  ((t (t 1+)) 0)  (((t t) 1+) 0)

where 1+ is a primitive procedure that adds 1 to its argument, and t is defined as follows:

(define (t f)
  (lambda (x) (f (f (f x)))))

Work this out yourself before you try it on the computer!