Quiz 10b

1. (2 points)
2. Louis Reasoner decides to implement his own version of streams. He writes the following code:

(define (cons-stream a b)

(cons a (delay b)))

(define(stream-car strm)

(car strm))

(define (stream-cdr strm)

(force (cdr strm)))

(define (integers-starting-from n)

(cons-stream n (integers-starting-from (+ n 1))))

He then types in (define ints (integers-starting-from 1)) and is shocked to find out that it doesn’t work. What happens? What is wrong with his implementation?

It infinite loops because cons-stream evaluates its arguments before delaying them, unlike the actual cons-stream, which is a special form. It tries to evaluate all the integers, which cannot happen because they are an infinite set.

1 pt for saying it infinite loops, 1 pt for reasoning.

2. Odd-squares

(define (parity-ints n)

(cons-stream n (parity-ints (+ n 2))))

(define odds (parity-ints 1))

(define squares (stream-map (lambda (x) (\* x x)) odds))

(define odd-squares (interleave odds squares))

OR

(define twos (cons-stream 2 twos)

(define odds (cons-stream 1 (add-streams odds twos))

;the following lines are the same as above

(define squares (stream-map (lambda (x) (\* x x)) odds))

(define odd-squares (interleave odds squares))

3. (3 points) Show the first 3 elements of each of these two streams :

(define (crazy x y)

(if (> x y)

x

(- x y)))

(define barney (cons-stream 1 (stream-map crazy barney stinson)))

(define stinson (cons-stream 2 (stream-map + barney stinson)))

barney: \_\_\_\_1\_\_\_\_ \_\_\_\_-1\_\_\_\_ \_\_\_\_-4\_\_\_\_

stinson: \_\_\_\_2\_\_\_\_ \_\_\_\_3\_\_\_\_ \_\_\_\_2\_\_\_\_

+0.5 for each element in the stream;

-1 for an arithmetic mistake or other trivial mistake that may or may not propagate.

4. Define the infinite stream updown where every element in the stream is a list (2 pt)

->(ss updown)

( (1) (1 2 1) (1 2 3 2 1) (1 2 3 4 3 2 1)

(1 2 3 4 5 4 3 2 1) . . . )

(define (updown-helper n)

(define (helper a)

(if (= a n)

(list a)

(append (list a) (helper (+ a 1)) (list a))))

(helper 1))

(define (updown-starting-from n)

(cons-stream (updown-helper n)

(updown-starting-from (+ n 1))))

(define updown

(updown-starting-from 1))

2 points for any working solution; some things to look for when deciding partial credit are the two helper procedures and the lack of a base case.