Quiz 1 Rubric

1. (5 points) In the spirit of new-if, we decide to write new-and.

(define (new-and x y z)

(and x y z))

1. (0.5 points each) How many times is + invoked in the following code:

(new-and (+ 1 1) #f (+ 2 2))

* In applicative order? \_\_\_\_2\_\_\_\_
* In normal order? \_\_\_\_1\_\_\_\_
* In Scheme? \_\_\_\_2\_\_\_\_

Note for grading: Give points for the third blank if a) the answer is 2 OR b) the answer is the same as the answer for applicative order.

1. (0.5 points each) What will Scheme output? If it is an error, write “Error”.

* (new-and 2 'foo (/ 1 0))

\_\_\_\_\_\_\_\_Error\_\_\_\_\_\_\_\_\_\_

* (and 2 'foo (/ 1 0))

\_\_\_\_\_\_\_\_Error\_\_\_\_\_\_\_\_\_\_

* (new-and 'neil 'patrick 'harris)

\_\_\_\_\_\_\_\_harris\_\_\_\_\_\_\_\_\_\_

1. (2 points) Are there any expressions where replacing and with new-and will change what happens? If so, give an example and explain why there is a difference.

Yes. There are two possible answers to this answer:

1) new-and must take 3 arguments, whereas and can take any number of arguments. So for example if you replace and with new-and in the expression

(and #t #f)

it will cause an error instead of returning #f

2) new-and is a procedure whereas and is a special form that may not evaluate all of its expressions. The difference is important in:

(and #f (/ 1 0))

Grading: 1 point for the example, 1 point for the explanation. For answers where they confuse and with or, for example (and #t (/ 1 0)), give 0.5 points for the example (and how many ever points the explanation is worth).

1. (5 points)
   1. (2 points) Write a procedure nth-letter that takes a word and a number n and returns the nth letter in that word. You can assume that n will be between 1 and the length of the word, inclusive.

>(nth-letter 'hello 1)

h

>(nth-letter 'hello 5)

o

(define (nth-letter wd n)

(if (= n 1)

(first wd)

(nth-letter (bf wd) (- n 1))))

Grading: 1 point for the recursive case, 1 point for the base case and the base case check. You can add or subtract 0.5 points as your fudge factor.

* 1. (3 points) Write a function nth-every using nth-letter that takes a sentence of words and sentence of numbers. nth-every will perform nth-letter on every member of the sentence of words using the corresponding number from the sentence of numbers and put them in a word. Assume both sentences will be the same length.

>(nth-every ‘(hi hello hola) ‘(1 1 1))

hhh

>(nth-every ‘(hi hello hola) ‘(1 2 4))

hea

(define (nth-every list1 list2)

(if (empty? list1)

“”

(word (nth-letter (first list1) (first list2))

(nth-every (bf list1) (bf list2)))))

Grading: Similar to previous problem. 1 point for base case, 1 point for call to nth-letter, 1 point for the recursive call, fudge factor of 0.5