1 Scheme

Give definitions for the following functions in Scheme:

1. The procedure *vowel* is a function of no arguments which returns a new vowel symbol each time it is called. The first time it is called *vowel* returns the symbol, *a*; the second time, *e*, etc. After being called five times, *vowel* returns the symbol *a* again, etc. For example,

    > (vowel)
    a
    > (vowel)
    e
    > (vowel)
    i
    > (vowel)
    o
    > (vowel)
    u
    > (vowel)
    a

2. The procedure *list->stream* takes a list as its argument and returns a stream. For example,

    > (list->stream '(a b c d))
    (a . #<promise>)

    Hint: Use *delay*.

3. The procedure *stream->list* takes a stream as its argument and returns a list. For example,

    > (stream->list (list->stream '(a b c d)))
    (a b c d)

    Hint: Use *force*.

4. The procedure *stream-map* works just like *map* except on streams. For example,

    > (stream->list (stream-map (lambda (x) (* x x))
                     (list->stream '(1 2 3 4))))
    (1 4 9 16)

    Hint: Minor modification of *map*. 


# 2 Haskell

Give definitions for following functions in Haskell:

1. The function `findIndices` takes a predicate and a list as arguments and returns a list of numbers indicating the positions of elements in the list which satisfy the predicate. For example,

   ```haskell
   *Main> findIndices (< 'a') "AbCdef"
   [0,2]
   *Main> findindices (== 0) [1,2,0,3,0]
   [2,4]
   ```

   The function `intersect` takes two lists as arguments and returns a list of elements common to both lists. For example

   ```haskell
   *Main> intersect "abc" "cat"
   "ac"
   *Main> intersect [1,2,3] [8]
   []
   ```

2. The function `isPrefixOf` takes two lists as argument and returns `True` iff the first list is a prefix of the second list. For example,

   ```haskell
   *Main> "foo" 'isPrefixOf' "foobar"
   True
   *Main> isPrefixOf [1,2,3] [4,5,6]
   False
   ```

3. The function `isSuffixOf` takes two lists as argument and returns `True` iff the first list is a suffix of the second list. For example,

   ```haskell
   *Main> "bar" 'isSuffixOf' "foorbar"
   True
   *Main> isSuffixOf [1,2,3] [4,5,6]
   False
   ```

4. The function `select` takes a predicate and two lists as arguments and returns a list composed of elements from the second list in those positions where the predicate, if applied to the element in the corresponding position of the first list, would return `True`.

   ```haskell
   *Main> :t select
   select :: (t -> Bool) -> [t] -> [a] -> [a]
*Main> select even [1..26] "abcdefghijklmnopqrstuvwxyz"
"bdfhjlnprtvxz"
*Main> select (<= 'g') "abcdefghijklmnopqrstuvwxyz" [1..26]
[1,2,3,4,5,6,7]

5. The function \texttt{zip3} takes three lists as arguments and returns a list of three-tuples. For example,

*Main> zip3 "01" [False,True] [0,1]
[('0',False,0),('1',True,1)]

6. The function \texttt{unzip3} takes a list of three-tuples as arguments and returns a three-tuple of lists. For example,

*Main> unzip3 [('l','b','a'),('o','f','f'),('l','f','k')]
("lol","bff","afk")
*Main> unzip3 (zip3 "lol" "bff" "afk")
("lol","bff","afk")