This tutorial will cover the following topics:

• Writing functions that produce and/or consume other functions.
• Practice with local definitions.
• Understanding the heap property.

For all problems in this tutorial, you may only define local helper functions.
Question 1: Line Functions
Write a function (make-line m b) which takes a slope m and an intercept b and produces a function that evaluates the line defined by m and b at the point x (where x is a parameter for the produced function). That is, the produced function should compute $y=mx+b$.

;;; Examples:
(define example-line (make-line 1.5 -5))
(check-expect (example-line 0) -5)
(check-expect (example-line 4.5) 1.75)
Question 2: Manipulating Strings
Write a function (str-change pred? func-change str) to perform custom string modifications. The function consumes a predicate (pred?) that determines which characters to modify and a change function (func-change) that converts characters in the string (str) to their new values.

For example, the code below uses str-change to convert all vowels in a string to uppercase.

(define (vowel? ch)  
  (member? ch (string->list "aeiou")))

;; Example:
(check-expect (str-change vowel? char-upcase "this is a test")  
  "thIs Is A tEst")
Question 3: Heap Property
In Assignment 8, you were introduced to a new type of binary tree called a heap. Every valid heap must satisfy the Heap Property: every key is less than or equal to every key in its subtrees. Here is the struct definition and data definition for a heap:

```scheme
(define-struct hnode (key left right))
;; A (heapof X) is one of:
;; * empty
;; * (make-hnode X (heapof X) (heapof X))
;; requires: all elements in left are >= key
;; all elements in right are >= key
```

Write a function (heap? bt) which determines whether a binary tree bt satisfies the heap property.

We use the following definitions for binary trees (you may assume that the comparison function <= is used when checking the heap property for this question. That is, the root’s key must be <= the keys of all elements in the left and right.):

```scheme
(define-struct node (key left right))
;; A Node is a (make-node Nat BT BT)
;; A binary tree (BT) is one of:
;; * empty
;; * Node
```
;; Examples:
(define example-heap (make-node 1 (make-node 2 empty empty)
                        (make-node 3 (make-node 4 empty empty)
                        (make-node 5 empty empty))))

(define not-a-heap (make-node 100 (make-node 2 empty empty)
                       (make-node 3 empty empty))

(check-expect (heap? example-heap) true)
(check-expect (heap? not-a-heap) false)