```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;; Structure and Interpretation of Computer Programs, 2. ed.      ;;;;;
;;;;; Section 1.1, Exercise 1.6                                      ;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;; Student: Abrantes Araújo Silva Filho                           ;;;;;
;;;;; Date: 2019-02-11                                               ;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


;;;; Alyssa P. Hacker doesn't see why if needs to be provided as a
;;;; special form. ``Why can't I just define it as an ordinary
;;;; procedure in terms of cond?'' she asks. Alyssa's friend Eva Lu
;;;; Ator claims this can indeed be done, and she defines a new
;;;; version of if:

(define (new-if predicate then-clause else-clause)
  (cond (predicate then-clause)
        (else else-clause)))

;;;; Eva demonstrates the program for Alyssa:

(new-if (= 2 3) 0 5)
; 5

(new-if (= 1 1) 0 5)
; 0

;;;; Delighted, Alyssa uses new-if to rewrite the square-root program:

(define (sqrt-iter guess x)
  (new-if (good-enough? guess x)
          guess
          (sqrt-iter (improve guess x) x)))

(define (average x y)
  (/ (+ x y) 2))

(define (improve guess x)
  (average guess (/ x guess)))

(define (good-enough? guess x)
  (< (abs (- (square guess) x)) 0.001))

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

(define (sqrt x)
  (sqrt-iter 1.0 x))

;;;; What happens when Alyssa attempts to use this to compute square
;;;; roots? Explain.

(sqrt 2)
; Aborting!: maximum recursion depth exceeded

; OK, so the procedure begins an infinite recursion and Scheme abort
; the procedure when it get into some predefined depth in
; recursion. What's going on? Let's try to evaluate:

; (sqrt-iter 1.0 2)
```

```scheme
; (new-if (good-enough? 1.0 2)
;       1.0
;       (sqrt-iter (improve 1.0 2) 2))

;(new-if (< (abs (- (square 1.0) 2)) 0.001)
;       1.0
;       (sqrt-iter 1.5 2))

;(new-if #f
;       1.0
;       (new-if (good-enough? 1.5 2)
;               1.5
;               (sqrt-iter (improve 1.5 2) 2)))

;(new-if #f
;       1.0
;       (new-if (< (abs (- (square 1.5) 2)) 0.001)
;               1.5
;               (sqrt-iter 1.333 2)))

;(new-if #f
;       1.0
;       (new-if #f
;               1.5
;               (new-if (good-enough? 1.333 2)
;                       1.333
;                       (sqrt-iter (improve 1.333 2) 2))))

;(new-if #f
;       1.0
;       (new-if #f
;               1.5
;               (new-if #f
;                       1.333
;                       (sqrt 1.4167 2))))

;(new-if #f
;       1.0
;       (new-if #f
;               1.5
;               (new-if #f
;                       1.333
;                       (new-if (good-enough? 1.4167 2)
;                               1.4167
;                               (sqrt-iter (improve 1.4167 2) 2)))))


; ... go on ... go on ... go on ... go on ... go on ... go on ...

; The problem here is (as explained on
; http://community.schemewiki.org/?sicp-ex-1.6) is that The default if
; statement is a special form which means that even when an
; interpreter follows applicative substitution, it only evaluates one
; of it's parameters- not both. However, the newly created "new-if"
; doesn't have this property and hence, it never stops calling itself
; due to the third parameter passed to it in sqrt-iter.
;
; So: this "new-if" IS NOT A SPECIAL FORM! The "new-if" is a NORMAL
; COMBINATION, so all the subexpressions are evaluated in the
; applicative-order substitution model: the interpreter needs to
; evaluate all subexpression. This results in an infinite
; recursion because the else-clause is always evaluated, EVEN IF THE
; "TEST" ("PREDICATE") WAS TRUE, because "new-if" IS NOT A SPECIAL
; FORM! If "new-if" was a special-form, when the predicate is true the
; else clause would not be evaluated; but this "new-if" is a NORMAL
; COMBINATION, so every subexpression IS EVALUATED BEFORE THE
; APPLICATION OF OPERATOR, so the else clause is infinitelly calling
; the procedure, and infinitelly goind depth in recursion. Note: the
; else clause of "new-if" will be evaluated EVEN IF THE "PREDICATE" is
; true... because of that, infinite recursion!
```