```scheme
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;; MIT 6.001 Fall/1997                                         ;;;;;
;;;;; Daniel Jackson - Recitation 1: Basics                        ;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;; Student: Abrantes Araújo Silva Filho                         ;;;;;
;;;;; Date: 2019-02-09                                             ;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;; Self-test: expressions, values & names

;;; Parens mean application
((* 4 5))
; Error in the evaluation phase because the expression evaluates to
; (20) wich aplies 20 to nothing

(* 4 5)
; 20


;;; Compound expressions
(+ (if (< 3 2) * 4) 7)
; 11

((if (< 3 2) + *) 4 5)
; 20


;;; Special & regular forms
(if (< 4 5) (+ 1 2) (5 5))
; 3. The expression (5 5) is wrong, but because if-expression is true,
; only the (+ 1 2) is evaluated.

(define (f a b) a)
; OK, this define a procedure named "f" that takes 2 arguments, "a"
; and "b", and returns "a"

(f (+ 1 2) (5 5))
; ERROR, because the f procedure is not a special form, so Scheme uses
; the general rule for evaluation, so (5 5) IS evaluated and is not
; possible do apply 5 to 5. The erros is in the evaluation phase.

(if if)
; ERROR, if-expression expects the predicate (test) and action.


;;; Naming
(define * 5)
; OK, but we are bounding the name "*" to the value of expression "5",
; wich is 5 itself. We loose the bound "*" -> "Multiplication Procedure"

(+ * 7)
; 12

(define 2 3)
; ERROR, could not redefine a self-evaluationg expression?


;;; No mutation of values
(define i 3)
; OK, just bound the name "i" to the value of "3" expression
```

```scheme
(define (inc n) (+ n 1))
; OK, just define the inc procedure wich adds 1 to argument

(inc i)
; 4

i
; 3. Scheme does not change the variable value


;;; No operators distinct from procedures
(if (< 2 3) 5 7)
; 5

(if (<> 2 3) 5 7)
; ERROR, there is no "<>" operator in Scheme

(define (<> i j) (not (= i j)))
; OK, just define the procedure "<>" wich retuns true if the arguments
; are different, and false if the arguments are egual

(if (<> 2 3) 5 7)
; 5. Note that the "<>" procedure could be used as an operator!


;;; No static types
(define (first x y) x)
; OK, just define the first-procedure wich return the first argument.

(first #f #t)
; #f

(first 3 4)
; 3

((first + -) 7 5)
; 12

((first + 3) 7 5)
;12


;;; Summary: essencial characteristics of Scheme
; a) Parens means application
; b) Compound expressions could be formed
; c) There is a general evaluation rule for expressions
; d) Special-forms have special evaluation rule
; e) Naming bind some symbol to some value, and the value could be a
;    procedure!
; f) There is NO mutation of values
; g) Operators are not distinct from procedures
; h) There is NO static types (type are bound to value, no to variable)!
```