More Scheme

Scheme

1. Basic Elements

(a) self-evaluating - expressions whose value is the same as the expression.

(b) names - Name is looked up in the symbol table to find the value associated with it. Names may be made of any collection of characters that doesn’t start with a number.

2. Combination

(value procedure arguments-separated-by-spaces)

Value is determined by evaluating the expression for the procedure and applying the resulting value to the value of the arguments.

3. Special Forms

(a) define - (define name value)

The name is bound to the result of evaluating the the value. Return value is unspecified.
(b) if - (if test consequent alternative)
If the value of the test is not false (#f), evaluate the consequent, otherwise evaluate the alternative.

(c) lambda - (lambda (arg1 ... argn) expression1 ... expressionn)
We will see this in more detail in lecture. A lambda captures a common pattern of computation as a procedure. When applied to a set of arguments, it “substitutes” the value of each expression for the corresponding argument in the body of the lambda, then evaluates the body.

Problems

1. Evaluate the following expressions

4
(+ 1 2)
(7)
(* (+ 7 8) (* - 5 6))
(define one 1)
(define two (+ 1 one))
(define five 3)
(+ five two)
(define biggie-size *)
(define hamburger 1)
(biggie-size hamburger five)

(= 7 (+ 3 4))

(= #t #f)

((+ 5 6))

2. Evaluate the following expressions (assuming x is bound to 3):

(if #t (+ 1 1) 17)

(if #f #f 42)

(if (> x 0) x (- x))

(if 0 1 2)

(if x 7 (7))

3. Evaluate the following expressions:

(lambda (x) x)

(((lambda (x) x) 17)

(((lambda (x y) x) 42 17)

(((lambda (x y) (y (/ 1 0)) 3)

(((lambda (x y) (x y 3)) (lambda (a b) (+ a b)) 14)

4. Suppose we’re designing an point-of-sale and order-tracking system for Wendy’s\(^1\). Luckily the Über-Qwuick drive through supports only 4 options: Classic Single Combo (hamburger with one patty), Classic Double With Cheese Combo (2 patties), and Classic Triple with Cheese Combo (3 patties), Avant-Garde Quadruple with Guacamole Combo (4 patties). We shall encode these combos as 1, 2, 3, and 4 respectively. Each meal can be biggie-sized to acquire a larger box of fries and drink. A biggie-sized combo is represented by 5, 6, 7, and 8 respectively.

(a) Write a procedure named biggie-size which when given a regular combo returns a biggie-sized version.

\(^1\) 16.001 and MIT do not endorse and are not affiliated with Wendy’s in any way. They merely capitalize on the pleasant way “biggie-size” rolls off the tongue.
(b) Write a procedure named \texttt{unbiggie-size} which when given a \textsl{biggie-sized} combo returns a non-\textsl{biggie-sized} version.

(c) Write a procedure named \texttt{biggie-size?} which when given a combo, returns true if the combo has been \textsl{biggie-sized} and false otherwise.

(d) Write a procedure named \texttt{combo-price} which takes a combo and returns the price of the combo. Each patty costs \$1.17, and a \textsl{biggie-sized} version costs \$0.50 extra overall.

(e) An order is a collection of combos. We’ll encode an order as each digit representing a combo. For example, the order 237 represents a Double, Triple, and \textsl{biggie-sized} Triple. Write a procedure named \texttt{empty-order} which takes no arguments and returns an empty order.

(f) Write a procedure named \texttt{add-to-order} which takes an order and a combo and returns a new order which contains the contents of the old order and the new combo. For example, \( \texttt{(add-to-order 1 2)} \rightarrow 12 \).

(g) Write a procedure named \texttt{order-size} which takes an order and returns the number of combos in the order. For example, \( \texttt{(order-size 237)} \rightarrow 3 \). You may find \texttt{quotient} (integer division) useful.

(h) Write a procedure named \texttt{order-cost} which takes an order and returns the total cost of all the combos. In addition to \texttt{quotient}, you may find \texttt{remainder} (computes remainder of division) useful.