1. What do the following Scheme expressions evaluate to?

(a) \((* 2 (+ 4 5))\) => 18
(b) \((= 3 (+ 1 3))\) => #f
(c) \((\text{car } \text{‘(elmer fudd daffy duck)})\) => elmer
(d) \((\text{cdr } \text{‘(elmer fudd daffy duck)})\) => (fudd daffy duck)
(e) \((\text{and } (= 1 2) (= 10 (/ 1 0)))\) => #f

2. Find the squid! For each of the following variables, write an expression that picks out the symbol squid. For example, for this definition: \((\text{define } x \text{ ‘(squid clam octopus)})\) the answer is \((\text{car } x)\).

(a) \((\text{define } y \text{ ‘(clam squid octopus)})\) => (cadr y)
(b) \((\text{define } z \text{ ‘(clam starfish (squid octopus) mollusc)})\) => (caaddr z)

3. Define a Scheme function to find the average of two numbers.

\(\text{(define (average } x y)\)
\(\quad (/ (+ x y) 2))\)

4. Define a Scheme function \(\text{mymax}\) to find the maximum of two numbers.

\(\text{(define (mymax } x y)\)
\(\quad (\text{if } (> x y) x y))\)

5. Suppose we evaluate the following Scheme expressions:

\((\text{define } x \text{ ‘(snail clam)})\)
\((\text{define } y \text{ ‘(octopus squid scallop)})\)

Draw box-and-arrow diagrams of the result of evaluating the following expressions. What parts of the list are created fresh, and which are shared with the variables \(x\) and \(y\)?

(a) \((\text{cons } \text{‘geoduck } x))\)
(b) \((\text{cons } y y)\)
(c) \((\text{append } x y)\)
(d) \((\text{cdr } y)\)

6. Define a recursive function \(\text{sum}\) to find the sum of the numbers in a list.

\(\text{(define (sum } s)\)
\(\quad (\text{if } (\text{null? } s)\)
\(\quad \quad 0\)
\(\quad \quad (+ (\text{car } s) (\text{sum } (\text{cdr } s))))))\)

7. Define a tail recursive version of \(\text{sum}\). (Define an auxiliary function if needed.)

\(\text{(define (sum } s)\)
\(\quad (\text{sum-helper } s 0))\)
\(\text{(define (sum-helper } s \text{sofar})\)
\(\quad (\text{if } (\text{null? } s)\)
\(\quad \quad \text{sofar}\)
\(\quad \quad (\text{sum-helper } (\text{cdr } s) (+ (\text{car } s) \text{sofar}))))\)

8. Define a recursive function \(\text{myfilter}\) like the built-in \(\text{filter}\) function in Scheme.
(define (myfilter p s)
  (cond ((null? s) ())
        ((p (car s)) (cons (car s) (myfilter p (cdr s))))
        (else (myfilter p (cdr s)))))

9. What is the result of evaluating the following Scheme expressions?

(a) (let ((x (+ 2 4))
          (y 100))
     (+ x y))

  106

(b) (let ((x 100)
          (y 5))
     (let ((x 1))
       (+ x y)))

  6

10. Define a function `mylength` to find the length of a list.

(define (mylength s)
  (if (null? s)
      0
      (+ 1 (mylength (cdr s))))