

1. Re-write the following expressions in Scheme and evaluate them using a Scheme interpreter/compiler.

- (a)  $(3 \times (5 + (10 \div 5)))$
- (b)  $(2 + 3 + 4 + 5)$
- (c)  $(1 + (5 + (2 + (10 \div 3))))$
- (d)  $(1 + (5 + (2 + (10 \div 3.0))))$
- (e)  $(3 + 5) \times (10 \div 2)$
- (f)  $(3 + 5) \times (10 \div 2) + (1 + (5 + (2 + (10 \div 3))))$

**Solution:**

- (a) `(* (+ (/ 10 5) 5) 3)`
- (b) `(+ 2 3 4 5)`
- (c) `(+ (+ (+ (+ (/ 10 3) 2) 5) 1)`
- (d) `(+ (+ (+ (+ (/ 10 3.0) 2) 5) 1)`
- (e) `(* (+ 3 5) (/ 10 2))`
- (f) `(+ (* (+ 3 5) (/ 10 2)) (+ (+ (+ (+ (/ 10 3) 2) 5) 1))`

2. Define a procedure `discount` that takes two arguments: an item's initial price and a percentage discount [1]. It should return the new price:

```
> (discount 10 5)
9.50
> (discount 29.90 50)
14.95
```

**Solution:**

```
(define (discount p d)
  (* p (- 1 (/ d 100.0))))
```

3. Write a function called `appearances` that returns the number of times its first argument appears as a member of its second argument [1].

**Solution:**

```
(define (appearances i l)
  (if (null? l)
      0
      (if (equal? i (car l))
          (+ 1 (appearances i (cdr l)))
          (appearances i (cdr l)))))
```

4. Write a procedure `inter` that takes two lists as arguments. It should return a list containing every element that appears in both lists, exactly once.

**Solution:**

```
(define (inter l1 l2)
  (if (null? l1)
      '()
      (if (and
            (memq (car l1) l2)
            (not (memq (car l1) (cdr l1))))
          (cons (car l1) (inter (cdr l1) l2))
          (inter (cdr l1) l2))))
```

5. Write a procedure `noatoms` that takes a list and returns the number of atoms it contains.

**Solution:**

```
(define (noatoms l)
  (if (null? l)
      0
      (if (not (or (pair? (car l)) (null? (car l))))
          (+ 1 (noatoms (cdr l)))
          (noatoms (cdr l)))))
```

6. Here is a Scheme procedure that never finishes its job:

```
(define (forever n)
  (if (= n 0)
      1
      (+ 1 (forever n))))
```

Explain why it doesn't give any result[1].

**Solution:** The terminating condition is: does  $n$  equal 0. However, each time `forever` is called,  $n$  is increased.

7. Write a function called `range` that takes an integer  $n$  and returns a list containing the atoms 1, 2, 3, ...,  $n$ .

**Solution:**

```
(define (range n)
  (if (= n 0)
      '()
      (append (range (- n 1)) (list n))))
```

8. Write a function called `reversel` that takes a list and returns it reversed.
9. If we list all the natural numbers below 10 that are multiples of 3 or 5, we get 3, 5, 6 and 9. The sum of these multiples is 23. Write a procedure to find the sum of all the multiples of 3 or 5 below 1000 [2].

**Solution:**

```
(define
  (sum35 n)
  (if (= 0 n)
      0
      (if (= 0 (modulo n 3))
          (+ n (sum35 (- n 1)))
          (if (= 0 (modulo n 5))
              (+ n (sum35 (- n 1)))
              (sum35 (- n 1))))))
```

10. Write a procedure called `flatten` that takes as its argument a list, possibly including sublists, but whose ultimate building blocks are atoms. It should return a sentence containing all the atoms of the list, in the order in which they appear in the original:

```
> (flatten '(((a b) c (d e)) (f g) (((h))) (i j) k)))
(a b c d e f g h i j k)
```

**Solution:**

```

(define (flatten l)
  (if (null? l)
      '()
      (if (pair? (car l))
          (append (flatten (car l)) (flatten (cdr l)))
          (cons (car l) (flatten (cdr l))))))

```

11. Each new term in the Fibonacci sequence is generated by adding the previous two terms. By starting with 1 and 2, the first 10 terms will be:

1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

By considering the terms in the Fibonacci sequence whose values do not exceed four million, find the sum of the even-valued terms [2].

**Solution:**

```

(define (sumevf n)
  (letrec (
    (fib
     (lambda (n)
       (if (= n 0)
           (list 0)
           (if (= n 1)
               (list 1 0)
               (let ((l (fib (- n 1))))
                 (cons (+ (car l) (cadr l)) l)))))))
    (apply
     +
     (map
      (lambda (x) (if (= 0 (modulo x 2)) x 0))
      (fib n)))))

; Bonus function: calculates the nth Fibonacci number.
(define (fib n)
  (if (= n 0)
      0
      (if (= n 1)
          1
          (fib (- n 1)))))

```

```

      (+ (fib (- n 1)) (fib (- n 2))))))

; Bonus function: lists the first n Fibonacci numbers.
(define (listfibs n)
  (letrec
    (
      (fib
       (lambda (n)
         (if (= n 0)
             (list 0)
             (if (= n 1)
                 (list 1 0)
                 (let ((l (fib (- n 1))))
                   (cons (+ (car l) (cadr l)) 1)))))))
    (fib n)))

```

12. Write a procedure `to-binary`:

```

> (to-binary 9)
1001
> (to-binary 23)
10111

```

**Solution:**

```

(define (binary n)
  (if (= n 0)
      '()
      (append
       (binary (/ (- n (modulo n 2)) 2))
       (list (modulo n 2)))))

```

13. Write Heap's algorithm for generating permutations in Scheme.

**Solution:**

```

; From: stackoverflow.com/questions/35869763
(define (generate n A)
  (cond

```

```
( (= n 1) (display A)
  (newline))
(else (let loop ((i 0))
        (generate (- n 1) A)
        (if (even? n)
            (swap A i (- n 1))
            (swap A 0 (- n 1)))
        (if (< i (- n 2))
            (loop (+ i 1))
            (generate (- n 1) A))))))

(define (swap A i1 i2)
  (let ((tmp (vector-ref A i1)))
    (vector-set! A i1 (vector-ref A i2))
    (vector-set! A i2 tmp)))

(define (heap l)
  (generate (length l) (list->vector l)))
```

## References

- [1] Brian Harvey and Matt Wright, *Simply Scheme: Introducing Computer Science*, MIT, 1999.
- [2] Project Euler, *Project Euler*, 2016.