The following exercises are related to the Racket programming language [3].

1. Re-write the following expressions in Racket and evaluate them using a Racket 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 [2]. 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. Define a function `grcomdiv` that takes two integers and returns their greatest common divisor.

```
> (grcomdiv 10 15)
5
> (grcomdiv 64 30)
2
```

Solution:

```
(define (grcomdiv a b)
  (if (< a b)
    (grcomdiv b a)
    (if (= 0 b)
      a
      (grcomdiv b (modulo a b)))))
```

4. Write a function called `appearances` that returns the number of times its first argument appears as a member of its second argument [2].

Solution:

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

5. 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:

```
; Use appearances from previous question.
(define (inter l1 l2)
  (if (null? l1)
    '()
    (if (> (appearances (car l1) l2) 0)
      (cons (car l1) (inter (cdr l1) l2))
      (inter (cdr l1) l2))))
```

6. 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)))))
```

7. Here is a Racket procedure that never finishes its job when n is not 0:

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

   Explain why it doesn't give any result[2].

   Solution: The terminating condition is: n equals 0. However, each time forever is called, n is increased.

8. 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))))
   ```

9. Write a function called `reversel` that takes a list and returns it reversed.

   Solution:

   ```
   (define (reversel l)
     (define (reversel-aux l a)
       (if (null? l)
         a
         (reversel-aux (cdr l) (cons (car l) a))))
     (reversel-aux l null))
   ```

10. 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 [1].

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))))))

; Call this with 999 for < 1000.
(sum35 999)
```

11. 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))))))
```

12. 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 [1].

Solution:

```
(define (sumflt total)
  (define (aux n-2 n-1 maxval total)
```

```
    (if (> (+ n-2 n-1) maxval)
      total
      (if (= 0 (modulo (+ n-2 n-1) 2))
        (aux n-1 (+ n-2 n-1) maxval (+ total n-2 n-1))
        (aux n-1 (+ n-2 n-1) maxval total))))
  (aux 0 1 total 0))
```

13. Write a procedure `to-binary` that takes a decimal interger and converts it into a list of 0's and 1's representing the number in binary form. The least significant bit should be on the right of the list.

```
> (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)))))
```

14. Write a function `select` that takes two elements, a list and a position in the list, and return the element of the list in that position.

```
> (select (list 1 2 3 4 5) 1)
2
```

Solution:

```
(define (select l i)
  (if (= 0 i)
      (car l)
      (select (cdr l) (- i 1))))
```

15. Write a function `perms` that takes a list as its only argument, and returns a list containing all permutations of that list.

Solution:

```
  ; From http://rosettacode.org/wiki/Permutations#Scheme
(define (insert l n e)
  (if (= 0 n)
      (cons e l)
      (cons (car l)
            (insert (cdr l) (- n 1) e))))

(define (seq start end)
  (if (= start end)
      (list end)
      (cons start (seq (+ start 1) end))))

(define (permute l)
  (if (null? l)
    '(())
    (apply append
      (map (lambda (p)
        (map
          (lambda (n) (insert p n (car l)))
          (seq 0 (length p))))
        (permute (cdr l))))))
```

References

[1] Project Euler. Project euler.

[2] Brian Harvey and Matt Wright. Simply scheme: Introducing computer science.

[3] PLT Inc. Racket – a programmable programming language.