Question 1. [17 marks]
Assume the following code has been run:

(require picturing-programs)

(define LLL
  (list 4 7 9 (list 35 28)
    (list (list 36 29) 30)))

Part (a) [4 marks]
Show the result of each expression: sample solution:

(length LLL)
5

(reverse LLL)
(list (list (list 36 29) 30) (list 35 28) 9 7 4)

(filter list? LLL)
(list (list 35 28) (list (list 36 29) 30))

(filter number? LLL)
(list 4 7 9)

Part (b) [9 marks]
Show two intermediate steps and the result for each expression. sample solution:

(map length (filter list? LLL))
(map length (list (list 35 28) (list (list 36 29) 30)))
(list (length(list 35 28)) (length (list (list 36 29) 30)))
(list 2 2)

(map reverse (filter list? LLL))
(map reverse (list (list 35 28) (list (list 36 29) 30)))
(list (reverse (list 35 28)) (reverse (list (list 36 29) 30)))
(list (list 28 35) (list 30 (list 36 29)))

(apply + (filter number? LLL))
(apply + (list 4 7 9))
(+ 4 7 9)
20

Part (c) [4 marks]
Show three intermediate steps and the result for the following expression: sample solution:
(apply append (filter list? (reverse LLL)))
(apply append (filter list? (list (list (list 36 29) 30) (list 35 28) 9 7 4)))
(apply append (list (list (list 36 29) 30) (list 35 28)))
(append (list (list 36 29) 30) (list 35 28))
(list (list 36 29) 30 35 28)

**Question 2.** [15 marks]

Assume the following code has been run:

(require picturing-programs)

**Part (a) [4 marks]**

Use the design check-expect below to write the definition for function double-stack, and to complete its type contract:

; design check-expect
(check-expect (double-stack (square 10 "outline" "black"))
  (above
    (scale 2 (square 10 "outline" "black"))
    (beside
      (square 10 "outline" "black")
      (square 10 "outline" "black")))))

**sample solution:**

; double-stack : image -> image
(define (double-stack an-image)
  (above
    (scale 2 an-image)
    (beside an-image an-image)))

**Part (b) [6 marks]**

Carefully read these check-expects for pyramid:

(check-expect (pyramid 0) \(\triangle\)) ; triangle of width 10

(check-expect (pyramid 1) \(\Delta\))

(check-expect (pyramid 2) \(\Delta\))
Complete the three check-expects below without drawing any images. You may use double-stack to help. Inside the check-expect for (pyramid 1) use the expression (pyramid 0). Inside the check-expect for (pyramid 2) use the expression (pyramid 1). sample solution:

(check-expect (pyramid 0) (triangle 10 "outline" "black"))

(check-expect (pyramid 1) (double-stack (pyramid 0)))

(check-expect (pyramid 2) (double-stack (pyramid 1)))

Part (c)  [5 MARKS]
Write the function pyramid: sample solution:

; pyramid : number -> image
(define (pyramid h)
  (cond
   [(= h 0) (triangle 10 "outline" "black")]
   [else (double-stack (pyramid (- h 1)))]))

Question 3.  [7 MARKS]
Assume function g below has been defined.

(define (g a-string)
  (cond
   [(< (string-length a-string) 2) a-string]
   [else
    (string-append
     (substring a-string 0 2)
     (g (substring a-string 2))
     (substring a-string 0 2))])))

Remember the behaviour of substring:

(check-expect (substring "abcde" 0 2) "ab")
(check-expect (substring "abcde" 2) "cde")

Part (a)  [4 MARKS]
Show the result of each of the following expressions: sample solution:

(g "ab")
"abab"

(g "abc")
"abcab"
Part (b) [3 marks]

Show at least two intermediate steps and the result of the following expression: sample solution:

(g "abcde")

(cond
  [(< (string-length a-string) 2) "abcde"
   [else
    (string-append
     (substring "abcde" 0 2)
     (g (substring "abcde" 2))
     (substring "abcde" 0 2))])]

(string-append
  "ab"
  (g "cde")
  "ab")

"abcdecdab"