```
;;=============================================================================;;
;; ANSI Common Lisp, Paul Graham, 1ª edição                                    ;;
;; http://www.paulgraham.com/acl.html                                          ;;
;;-----------------------------------------------------------------------------;;
;; Exercícios do Capítulo: 2                                                   ;;
;;-----------------------------------------------------------------------------;;
;; Por: Abrantes Araújo Silva Filho                                            ;;
;;      abrantesasf@pm.me                                                      ;;
;;=============================================================================;;

;; Exercício 1:
(+ (- 5 1) (+ 3 7))
                                        ; 14

(list 1 (+ 2 3))
                                        ; (1 5)

(if (listp 1)
    (+ 1 2)
    (+ 3 4))
                                        ; 7

(list
 (and (listp 3) t)
 (+ 1 2))
                                        ; (nil 3)


;; Exercício 2:
(cons 'a '(b c))

(cons 'a (cons 'b '(c)))

(cons 'a (cons 'b (cons 'c '())))


;; Exercício 3:
(defun quarto-elemento (lst)
  (car (cdr (cdr (cdr lst)))))

(quarto-elemento '(1 2 3 4 5))


;; Exercício 4:
(defun maior (x y)
  (if (> x y)
      x
      y))

(maior 2 3)
(maior 3 2)
(maior 5 5)


;; Exercício 5:
(defun enigma (x)
  (and (not (null x))
       (or (null (car x))
           (enigma (cdr x)))))
```

```
(enigma '(a b c))
(enigma '())
(enigma '(a))

(defun mystery (x y)
  (if (null y)
      nil
      (if (eql (car y) x)
          0
          (let ((z (mystery x (cdr y))))
            (and z (+ z 1))))))

(mystery 1 '(5 4 3 2 1))
(mystery 0 '(5 4 3 2 1))
(mystery 3 '(3 1))
(mystery 3 '(1 3))
(mystery 3 '(1 2 3))
(mystery 3 '(1 2 4 3))


;; Exercício 6:
(car (car (cdr '(a (b c) d))))
                                        ; B

(or 13 (/ 1 0))
                                        ; 13

(apply #'list 1 nil)
                                        ; (1)

(apply #'+ 1 2 3 nil)
                                        ; 6


;; Exercício 7:
(defun contem-lista? (lst)
  (if (null lst)
      nil
      (if (listp (car lst))
          t
          (contem-lista? (cdr lst)))))

(contem-lista? '(1 2 3))
(contem-lista? '(1 2 3 (4 5) 6 7))
(contem-lista? '(nil))
(contem-lista? '(()))


;; Exercício 8:
(defun imprime-dots-iter (n)
  (do ((i 1 (+ i 1)))
      ((> i n) nil)
    (format t "."))
  (format t "~%"))
(imprime-dots-iter 10)

(defun imprime-dots-rec (n)
  (if (eql n 0)
      nil
      (progn
```

```
        (format t ".")
        (imprime-dots-rec (- n 1)))))
(imprime-dots-rec 10)
(imprime-dots-rec 3)

(defun conta (x lst)
  (let ((n 0))
    (dolist (i lst)
      (if (eql i x)
          (setf n (+ n 1))))
    n))
(conta 'a '(b a n a n a))
(conta 'b '(b a n a n a))
(conta 'n '(b a n a n a))
(conta 'z '(b a n a n a))

(defun conta-rec (x lst)
  (let ((z 0))
    (if (null lst)
        nil
        (if (eql x (car lst))
            (setf z (+ z 1))
            (conta-rec x (cdr lst))))
    z))

(conta-rec 'a '(b a n a n a))
(conta-rec 'b '(b a n a n a))
(conta-rec 'n '(b a n a n a))
(conta-rec 'z '(b a n a n a))


;; Exercício 9:
(defun summit (lst)
  (setf lst (remove nil lst))
  (apply #'+ lst))

(summit '(1 nil 2 nil 3 nil 4 nil 5))

(defun summit2 (lst)
  (let ((x (car lst)))
    (if (null x)
        (summit2 (cdr lst))
        (+ x (summit2 (cdr lst))))))

(summit2 '(1 nil 2 nil 3 nil 4 nil 5))
```