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


;;; Seção 2.1: Forma de expressões Lisp
1

(+ 2 3)

(+ 2 3 4)

(+)

(+ 2)

(+ 2 3 4 5)

(/ (- 7 1)
   (- 4 2))


;;; Seção 2.2: Avaliação de expressões
(/ (- 7 1)
   (- 4 2))

(/ 1 0)

(quote (+ 3 5))

'(+ 3 5)


;;; Seção 2.3: Dados
"ora et labora"

'Artichoke

'(my 3 "Sons")

'(the list (a b c) has 3 elements)

(quote (the list (a b c) has 3 elements))

(list 'my (+ 2 1) "Sons")

(list '(+ 1 2) (+ 1 2))

(list '(+ 1 2) (+ 1 2) 4 "DiFeReNtE" '(1 2 3 4 5))

()
'()

nil
```

```
'nil

(list ())

(list '())

(list () '() nil 'nil)


;; Seção 2.4: Operações em listas
(cons 'a nil)

(cons 'a (cons 'b nil))
(list 'a 'b)

(car '(a b c))
(car ())
(car '())

(cdr '(a b c))
(cdr ())

(car (cdr (cdr '(a b c d))))
(third '(a b c d))

(car (cdr (cdr '(a b))))
(third '(a b))


;; Seção 2.5: Verdade
t
nil

(listp '(a b c))

(listp 4)

(listp nil)

(null '(a))

(null 4)

(null ())

(not nil)

(not (null ()))

(not (null '(nil)))
(not (null '(())))

(if (listp '(a b c))
    "Sim"
    "Não")

(if (listp 10)
    "Sim"
    "Não")
```

```
(if "Verdade"
    "Sim"
    "Não")

(or t nil)
(or nil t)
(or 1 nil)
(or nil 1)
(or nil nil "retorna aqui" "mas não chega aqui")

(and t t t t nil)
(and 1 2 3 4 5)
(and "a" 'nome)
(and 1 2 nil 4 5)


;; Seção 2.6: Funções
(defun terceiro (l)
  (car (cdr (cdr l))))

(defun soma-maior (x y z)
  (> (+ x y)
     z))


;; Seção 2.7: Recursão
(defun our-member (obj lst)
  (if (null lst)
      nil
      (if (eql (car lst)
               obj)
          t
          (our-member obj (cdr lst)))))


;; Seção 2.8: Lendo Lisp


;; Seção 2.9: Input e Output
(format t "~A mais ~A é igual a ~A.~%" 2 3 (+ 2 3))

(defun pergunte (s)
  (format t "~A" s)
  (read))


;; Seção 2.10: Variáveis
(let ((x 1)
      (y 2))
  (+ x y))

(defun pergunte-numero ()
  (format t "Informe um número: ")
  (let ((n (read)))
    (if (numberp n)
        n
        (pergunte-numero))))

(defun pergunte-idade ()
  (format t "Qual sua idade (informe um número): ")
```

```
  (let ((idade (read)))
    (if (numberp idade)
        idade
        (pergunte-idade))))

(defparameter *glob* 99)
*glob*
(format t "A variável global *glob* vale: ~A~%" *glob*)

(defconstant limite 100)
limite

(let ((*glob* 1))
  *glob*)
*glob*

(let ((limite 1))
  limite)          ; erro, pois LIMITE é constante
limite

(boundp '*glob*)
(boundp (quote *glob*))

(boundp 'limite)
(boundp (quote limite))


;; Seção 2.11: Alocação
*glob*
(setf *glob* 50)
*glob*

(let ((*glob* 10))
  *glob*)
*glob*

(let ((x 1))
  (setf x 2)
  x)

(setf limite 0) ; erro, pois LIMITE é constante

(defparameter *lista* '(a b c d e))
*lista*

(setf (car *lista*) 'z)
*lista*

(setf (cdr *lista*) 'a)
*lista*

(defparameter *lista* '(a b c d e))
*lista*

(setf (third *lista*) 'z)
*lista*


;; Seção 2.12: Programação Funcional
; Significa escrever programas através de funções que RETORNAM VALORES, ao invés
```

```
; de funções que MODIFICAM COISAS. Tentamos evitar ao máximo usar funções que
; modificam coisas, como SETF ou outras.

(defparameter *lista* '(b a n a n a))
*lista*

(remove 'a *lista*)
*lista*

(setf *lista* (remove 'a *lista*))
*lista*


;; Seção 2.13: Iteração
(defun mostra-quadrados (inicio fim)
  (do ((i inicio (incf i)))
      ((> i fim) 'fim)
    (format t "~A x ~A = ~A~%" i i (* i i))))
(mostra-quadrados 0 10)

;; Versão recursiva da função acima:
(defun mostra-quadrados-rec (inicio fim)
  (if (> inicio fim)
      'fim
      (progn
        (format t "~A x ~A = ~A~%" inicio inicio (* inicio inicio))
        (mostra-quadrados-rec (+ inicio 1) fim))))
(mostra-quadrados-rec 0 10)

(defun tamanho (lst)
  (let ((len 0))
    (dolist (i lst)
      (setf len (+ len 1)))
    len))
(tamanho '(a b c d e))

;; Versão recursiva da função acima
(defun tamanho-rec (lst)
  (if (null lst)
      0
      (+ 1 (tamanho-rec (cdr lst)))))
(tamanho-rec '(a b c d e f))

(defun imprime-lista (lst)
  (dolist (i lst)
    (format t "~A~%" i))
  (format t "~%"))
(imprime-lista '(a b c d e))

;; Versão recursiva da função acima
(defun imprime-lista-rec (lst)
  (if (null lst)
      nil
      (format t "~A~%" (car lst)))
  (imprime-lista-rec (cdr lst)))
(imprime-lista-rec '(a b c d e))


;; Seção 2.14: Funções como Objetos
(function +)
```

```
#'+

(apply (function +) '(1 2 3 4 5))
(apply #'+ '(1 2 3 4 5))

(apply (function -) '(100 2 3 4 5 6))
(apply (function -) 100 2 3 4 5 '(6)) ; o último argumento tem que ser uma lista

(funcall #'* 5 4 3 2 1) ; o último argumento não precisa ser uma lista

((lambda (x)     ; aqui começa a definição de uma função SEM NOME
   (+ 100 x))
 1)              ; este é o argumento passado à função

(funcall #'(lambda (x)
             (+ 1 x))
         100)


;; Seção 2.15: Tipos
;; Os VALORES é que têm tipos, as variáveis não. Isso é chamado de
;; MANIFEST TYPING.
;; Todo valor tem mais de um tipo

(typep 27 'fixnum)
(typep 27 'integer)
(typep 27 'rational)
(typep 27 'real)
(typep 27 'number)
(typep 27 'atom)
(typep 27 't)
```