```
;;;;============================================================================;;;;
;;;; Common Lisp: A Gentle Introduction to Symbolic Computation                 ;;;;
;;;; http://www.paulgraham.com/acl.html                                         ;;;;
;;;;----------------------------------------------------------------------------;;;;
;;;; Exercícios: 1                                                              ;;;;
;;;;----------------------------------------------------------------------------;;;;
;;;; Por: Abrantes Araújo Silva Filho                                           ;;;;
;;;;      abrantesasf@pm.me                                                     ;;;;
;;;;============================================================================;;;;

;;; Exercício 1.1:
;;; --------------

(+ 6 7)
(* 3 4)
(/ 16 8)
(- 4 3)
(abs -3)
(* -8 6)
(/ 15 9)
(+ 8 0)
(+ 8) ; evitar usar
(- 5 6)
(- 1 1/3)
(abs (+ -5 3))


;;; Exercício 1.2:
;;; --------------

;S   AARDVARK
;I   87
;S   PLUMBING
;S   1-2-3-GO
;I   1492
;N   3.14159265358979
;N   22/7
;S   zerop
;I   0
;I   -12
;S   SEVENTEEN


;;; Exerício 1.3:
;;; -------------

(< 7 11)
(oddp 12)
(equal 'kirk 'spock)
(numberp 12)
(numberp 'twelve)
(< -4 -3)
(zerop 0)
(equal 9 -9)
(equal 9 (abs -9))


;;; Exercício 1.4:
;;; --------------
```

```
(defun sub2 (x)
  (- x 2))

(sub2 3)


;;; Exercício 1.5:
;;; --------------

(defun twop (x)
  (zerop (sub2 x)))

(twop 3)
(twop 2)


;;; Exercício 1.6:
;;; --------------

(defun half (x)
  (/ x 2))

(half 1)

(defun half (x)
  (- x (/ x 2)))

(half 3)


;;; Exercício 1.7:
;;; --------------

(defun multi-digit-p (x)
  (> x 9))

(multi-digit-p 8)
(multi-digit-p 10)


;;; Exercício 1.8:
;;; --------------

(defun negativa (x)
  (- 0 x))

(negativa 3)


;;; Exercício 1.9:
;;; --------------

(defun twomorep (x y)
  (equal x (add2 y)))

(twomorep 4 2)
(twomorep 7 3)


;;; Exercício 1.10:
```

```
;;; ---------------

(defun twomorep (x y)
  (equal (sub2 x) y))

(twomorep 4 2)
(twomorep 7 3)


;;; Exercício 1.11:
;;; ---------------

(defun average (x y)
  (+ (/ x 2)
     (/ y 2)))

(average 4 6)
(average 6 7)


;;; Exercício 1.12:
;;; ---------------

(defun more-than-half-p (x y)
  (> x (/ y 2)))

(more-than-half-p 3 2)
(more-than-half-p 4 10)


;;; Exercício 1.13:
;;; ---------------

(defun same-result (x)
  (symbolp (numberp x)))

(same-result 20)
(same-result 'a)


;;; Exercício 1.14:
;;; ---------------

(not nil)
(not 12)
(not 'not)


;;; Exercício 1.15:
;;; ---------------

(defun not-onep (x)
  (not (equal x 1)))

(not-onep 1)
(not-onep 99)


;;; Exercício 1.16:
;;; ---------------
```

```
(defun not-plusp (x)
  (not (> x 0)))

(not-plusp 0)
(not-plusp 2)


;;; Exercício 1.17:
;;; ---------------

(defun my-evenp (x)
  (not (oddp x)))

(my-evenp 1)
(my-evenp 2)


;;; Exercício 1.18:
;;; ---------------

(defun my-crazy (x)
  (zerop (add1 (add1 x))))

(my-crazy -2)
(my-crazy 0)


;;; Exercício 1.19:
;;; ---------------

(defun dupla-negacao (x)
  (not (not x)))

(dupla-negacao 'nil)
(dupla-negacao t)
(dupla-negacao 'rutabaga)


;;; Exercício 1.20:
;;; ---------------

(defun my-xor (x y)
  (not (equal x y)))

(my-xor t 'nil)
(my-xor 'nil t)
(my-xor t t)
(my-xor 'nil 'nil)
(my-xor 'casa 'banana)
(my-xor 'casa 'casa)


;;; Exercício 1.21:
;;; ---------------

(defun test1 (x)
  (add1 (zerop x)))
;; tenta adicionar 1 à um booleano
```

```
(defun test2 (x y)
  (equal (+ x y)))
;; equal será chamado com apenas um argumento

(defun test3 (x y)
  (symbolp (not x y)))
;; not está sendo chamado com mais de um argumento


;;; Exercício 1.22:
;;; ---------------

;; Sim, todos os predicados são funções que retornam T ou NIL.


;;; Exercício 1.23:
;;; ---------------

;; not, equal, <, >


;;; Exercício 1.24:
;;; ---------------

;; Sim e sim.


;;; Exercício 1.25:
;;; ---------------

;;; Porque a única maneira de indicar falso/não em Lisp é com o
;;; símbolo NIL --- ou nil ou ().


;;; Exercício 1.26:
;;; ---------------

;; a) Falso, existem predicados que aceitam valores diferentes de T ou NIL
;; b) Verdadeiro, todos os predicados produzem T ou NIL como output


;;; Exercício 1.27:
;;; ---------------

(evenp 'casa)
(evenp 2 3)
```