```
"""#-------------------------------------------------------------------------------#"""
"""#----------------------       METU Cognitive Sciences       ----------------------#"""
"""#----------------------          Turgay Yıldız              ----------------------#"""
"""#----------------------      yildiz.turgay@metu.edu.tr      ----------------------#"""
"""#-------------------------------------------------------------------------------#"""



"""-------------------------------------------------------------------------------"""
"""------------------                   Exercise 2.1                  ------------------"""
"""-------------------------------------------------------------------------------"""

; Define a procedure named ASCENDINGP that takes three numbers as input and re-
; turns T if the numbers are in ascending order, and NIL otherwise. Equality means
; ascension, therefore (ASCENDINGP 3 4 4) must return T.

(defun ascendingp (x y z)

  (and (<= x y) (<= y z))
)


"""-------------------------------------------------------------------------------"""
"""------------------                   Exercise 2.2                  ------------------"""
"""-------------------------------------------------------------------------------"""

; Define a procedure that takes two numbers and returns -1 if their difference is
; negative, 0 if they are equal, and 1 if their difference is positive. You do not need
; to check for numberhood, assume that the user will always give numbers as input.
; You are allowed to compute the difference of the input numbers only once; and
; SETF and DEFVAR are forbidden.



(defun neg-eq-pos (x y)

  (if   (< (- x y) 0 )    ; or (<  x  y)
  (- 0 1)

  (if (= x y)
      0
      1
      )
  )
  )

"""-------------------------------------------------------------------------------"""
"""------------------                   Exercise 2.3                  ------------------"""
"""-------------------------------------------------------------------------------"""
; Solve Ex. 2.2, this time by checking for numberhood as well. Your program should
; return NIL if any (or both) of the numbers is not a number. Do NOT use AND.

(defun func1 (x y)

  (if (or (not (numberp x))    (not (numberp y))  )
      nil
      (+ x y)
      )
  )


"""-------------------------------------------------------------------------------"""
"""------------------                   Exercise 2.4                  ------------------"""
"""-------------------------------------------------------------------------------"""

; Define a procedure that takes three numbers and returns T if all the three are integers
; and returns NIL otherwise. Do NOT use AND.

(defun func2 (x y z)

  (or  (not (integerp x))  (not (integerp y))   (not (integerp z)) )
  nil
  t
  )
```

```
"""------------------------------------------------------------------------------------------"""
"""------------------                    Exercise 2.5                    ------------------"""
"""------------------------------------------------------------------------------------------"""

(defun howcompute (x y z)

  (cond

   ((=  (- x y) z) (print "subtracted"))
   ((=  (- y z) z) (print "subtracted"))
   ((=  (+ x y) z) (print "add"       ))
   ((=  (* x y) z) (print "multiplied"))
    (t             (print "dont know" ))
    )
  )


"""------------------------------------------------------------------------------------------"""
"""------------------                    Exercise 2.6                    ------------------"""
"""------------------------------------------------------------------------------------------"""

(defun func3 (x y)

  (if (and (numberp x)  (numberp y) )
      (if (>= x y)
    x
    y
    )
      nil
      )
  )

"""------------------------------------------------------------------------------------------"""
"""------------------                    Exercise 2.7                    ------------------"""
"""------------------------------------------------------------------------------------------"""

(defun func4 (x y z)

  (func3 (func3 x y) z)
  )


"""------------------------------------------------------------------------------------------"""
"""------------------                    Exercise 2.8                    ------------------"""
"""------------------------------------------------------------------------------------------"""

(defun func5  (x y z)

  (if (and (<= x y) (<= x z) )
      (if (<= y z)
    y
    z
    )
      (if (and (>= x y) (>= x z) )
    (if (>= y z)
        y
        z
        )
    x
    )
      )
  )



"""------------------------------------------------------------------------------------------"""
"""------------------                    Exercise 2.11                   ------------------"""
"""------------------------------------------------------------------------------------------"""

(defun halver (x)

  (if (< x 1)
      x
      (let  ( (h (/ x 2)) )
       ( halver h)
```

```
      )
  )
)

(defun halver2 (x)

  (if (<  x  1)
    x
    (halver (/ x 2) )
    )
    )


"""----------------------------------------------------------------------------------------"""
"""------------------                   Exercise 2.12                   ------------------"""
"""----------------------------------------------------------------------------------------"""

; Rewrite (AND X Y Z W) by using cond COND.11

(defun func12 (x y z w)

  (cond
  (X
   (cond
     (Y
      (cond
        (Z
         (cond
           (W t)
           (t nil)))
       (t nil)))
   (t nil)))
  (t nil))
  )


"""----------------------------------------------------------------------------------------"""
"""------------------                   Exercise 2.13                   ------------------"""
"""----------------------------------------------------------------------------------------"""

; Write COND statements equivalent to: (NOT U) and (OR X Y Z)

(defun myfunc13 (x)

  (cond (x   (not x) )
        (t      t    )
  )
  )

; (OR X Y Z)

(defun myfunc13-2 (x y z)

  (cond
    (X t)
    (Y t)
    (Z t))
  )


(defun myfunc13-3 (x y z)

  (cond
    (x  x)
    (y  y)
    (z  z)
    (t  nil)
  )
)
```