```
"""#-----------------------------------------------------------------------------#"""
"""#----------------------        METU Cognitive Sciences        ----------------------#"""
"""#----------------------         Symbols & Programming         ----------------------#"""
"""#----------------------             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 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
      (neg_eq_pos x y)
      )
  )

; (func1 2 'a)    ->    nil
; (func1 2  a)    ->    error

"""-----------------------------------------------------------------------------"""
"""------------------                  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)

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


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

```
; Write a function HOWCOMPUTE taking 3 numbers, telling the basic arithmetic oper-
; ation that is used to compute the third number from the first two – it should say
; so if it cannot find it. Your response can be one of ADDED, MULTIPLIED, DIVIDED,
; SUBTRACTED, DONT-KNOW.9 . Use COND in your answer.

(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                  -------------------"""
"""----------------------------------------------------------------------------------------"""

; Define a function that takes two arguments and returns the greater of the two.

(defun func3 (x y)

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

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

; Define a procedure that takes three arguments and returns the greatest of the three.

(defun func4 (x y z)

    (func3 (func3 x y) z)
)


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

; Define a procedure that takes three numbers and gives back the second largest of
; them. Use only IF and comparison predicates like <, <=, etc.

(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)))

"""-------------------                  Second Way                  -------------------"""

(defun func5_2 (x y z)

  (cond   ( (and (<= x y) (<= x z))               (if (<= y z)  y   z))
          ( (and (>= x y) (>= x z))               (if (>= y z)  y   z))
          ( t                                      x)
  ))
```

```
"""----------------------------------------------------------------------------------------"""
"""------------------                    Exercise 2.9                    ------------------"""
"""----------------------------------------------------------------------------------------"""

; Define a procedure that takes three numbers and gives back the sum of the squares
; of the larger two.

(defun sos (x y)  (+ (expt x 2) (expt y 2)))

(defun func9 (x y z)

  (cond ((and (<= x y) (<= x z))               (sos y z))
        ((and (<= y x) (<= y z))               (sos x z))
        (t                                     (sos x y))
  ))
"""----------------------------------------------------------------------------------------"""
"""------------------                    Exercise 2.10                   ------------------"""
"""----------------------------------------------------------------------------------------"""

; Solve Ex. 2.8 using COND.

(defun func5_2 (x y z)

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

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

; Define a procedure named halver that halves a given number until the result be-
; comes less than 1 and returns that result – solve the problem by making your pro-
; cedure call itself. Here is an example interaction with such a procedure:

(defun halver (x)

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


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

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

(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)


; (NOT U)
```

```
(defun myfunc13 (x)

  (cond (x   (not x) )  ; when x == True, if x == NIL , then this does not work
        (t      t    )  ; otherwise, if you remove this line and if x is NIL, output will be NIL
  ))

; (OR X Y Z)

(defun myfunc13-2 (x y z)

  (cond
    (X t)
    (Y t)
    (Z t))
  )         ; otherwise NIL, you dont need to add the line :   (t            nil)


(defun myfunc13-3 (x y z)

  (cond
    (x  x)
    (y  y)
    (z  z)
  )
)

"""-------------------------------------------------------------------------------------------"""
"""------------------                   Exercise 2.14                   ------------------"""
"""-------------------------------------------------------------------------------------------"""

; Write the final version of the CHANGE-COND program using only AND and OR, no IF, no COND.

( defun changer-cond ( n )

    ( cond  (( not ( numberp n ))               nil )
            (( not ( integerp n ))              ( changer-cond ( round n )))
            (( zerop ( rem n 3))                (+ (* 3 n ) 1))
            ( t                                 n )))



(defun func14 (n)

  (and (numberp n)

  (or
    (and (not ( integerp n ))               (changer-cond ( round n )))
    (and (zerop ( rem n 3))                 (+ (* 3 n ) 1))
    n)))


"""-------------------------------------------------------------------------------------------"""
"""------------------                   Exercise 2.15                   ------------------"""
"""-------------------------------------------------------------------------------------------"""

; The following definition is meant to mimic the behavior of IF using AND and OR.

( defun custom-if ( test succ fail )

        ( or ( and test succ ) fail ))


; But it is unsatisfactory in one case, what is it? Define a better procedure which
; avoids this failure.

; problem is * (custom-if T nil T)     returns   T.

( defun func15 ( test succ fail )

    (or (and test succ)  (and (not test) fail) ))
```