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

"""-----------------------------------------------------------------------------------------"""
"""-------------------                   Exercise 8.1                   -------------------"""
"""-----------------------------------------------------------------------------------------"""

Define a procedure APPEND2 that appends two lists.

(a b c) (1 2 3)      ->     (a b c 1 2 3)

(defun append2 (lst1 lst2)

    (dolist (i (reverse lst1) lst2)

        (setf lst2 (cons i lst2))
    )
)

* (append2 '(a b c) '(1 2 3) )

(A B C 1 2 3)

"""-----------------------------------------------------------------------------------------"""
"""-------------------                   Exercise 8.2                   -------------------"""
"""-----------------------------------------------------------------------------------------"""

Define an iterative procedure CHOP-LAST, which removes the final element of the
given list – its like CDR from the back. You are NOT allowed to make
(REVERSE (CDR (REVERSE LST))). Nothing to be done for an empty list, just
return it as it is; but a single element list gets “nilled”.

(a b c d)   ->    (a b c)

(defun  chop_last   (x   storage)

    (cond   ( (endp x)                              nil)
            ( (endp (cdr x))                        (reverse storage))
            ( t                                     (chop_last (cdr x)  (cons (car x) storage)))
        )
)

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

(defun chop_last2 (lst)

    (mapcar #' (lambda (x y)

        x)

    lst (make-list (- (length lst) 1)))

)

"""-----------------------------------------------------------------------------------------"""
"""-------------------                   Exercise 8.3                   -------------------"""
"""-----------------------------------------------------------------------------------------"""


Define an iterative procedure UNIQ that takes a list and removes all the repeated
elements in the list keeping only the first occurrence. This is the expected behavior:

* (uniq '(a b r a c a d a b r a))

    (A B R C D)


(defun unique-list  (lst)   ; initially  (storage = nil) and (reader = (car x) )

    (cond   ( (null lst)                        nil)
            ( (member (car lst) (cdr lst))      (unique-list (cdr lst)))
            ( t                                 (cons (car lst) (unique-list (cdr lst))))
    )
)


"""-----------------------------------------------------------------------------------------"""
"""-------------------                   Exercise 8.4                   -------------------"""
"""-----------------------------------------------------------------------------------------"""

Define a procedure that reverses the top-level elements of a list.

( a b c (d e f) g k l)   ->    (l k g (d e f) c b a)


(defun my_reverse (lst storage)

    (cond   ( (null lst)                            storage)
            ( t                                     (my_reverse (cdr lst) (cons (car lst) storage)))
    )
)


"""-----------------------------------------------------------------------------------------"""
"""-------------------                   Exercise 8.5                   -------------------"""
"""-----------------------------------------------------------------------------------------"""

The mean of n numbers is computed by dividing their sum by n. A running mean
is a mean that gets updated as we encounter more numbers. Observe the following
input-output sequences:
```

```
* (run-mean '(3 5 7 9))

            (3 4 5 6)

The first element 3 is the mean of the list (3), the second element 4 is the mean
of (3 5), and so on. Implement RUN-MEAN by using DOTIMES and NTH.


(defun run_mean (lst)

    (let    ( (result                           nil)
              (cum_total                        0)
              (mean                             0)
            )
        (dotimes    (i (length lst) (reverse result))

            (setf cum_total   (+ (nth i lst) cum_total))
            (setf mean  (float (/ cum_total (+ i 1))))
            (setf result (cons mean result))
        )
    )
)

"""----------------------------------------------------------------------------------------"""
"""-------------------                  Exercise 8.6                  -------------------"""
"""----------------------------------------------------------------------------------------"""


Define a procedure SEARCH-POS that takes a list as search item, another list as a
search list and returns the list of positions that the search item is found in the search
list. As usual, positioning starts with 0. Use DOTIMES. A sample interaction:


* (search-pos '(a b) '(a b c d a b a b))
                        (6 4 0)

* (search-pos '(a a) '(a a a a b a b))
                        '(2 1 0)

(defun search_pos (lst1 lst2 counter storage)

    (let    ( (result                   t)
              (flag                     0)
              (len_1                    (length lst1))
              (len_2                    (length lst2))
            )

        (cond   ( (> len_1 len_2)                       (reverse storage))
                  (t

                    (if

                        (dotimes    (i len_1 result)

                            (if     (and (equal (nth i lst1) (nth i lst2)) (= flag 0))
                                    (setf result t)
                                    (and (setf result nil) (setf flag 1))
                            )
                        )
                        (search_pos lst1 (cdr lst2) (+ counter 1) (cons counter storage))
                        (search_pos lst1 (cdr lst2) (+ counter 1) storage)
                    )
                )
        )
    )
)


"""----------------------------------------------------------------------------------------"""
"""-------------------                  Exercise 8.7                  -------------------"""
"""----------------------------------------------------------------------------------------"""

Define a procedure that reverses the elements in a list including its sublists as well.


( a b c (d e f) g k l)   ->    (l k g (f e g) c b a)


(defun my_reverse2 (lst storage)

    (cond   ( (null lst)                            storage)
            ( (listp (car lst))                     (append (my_reverse2 (cdr lst) nil)  (list (my_reverse2 (car lst) nil)) storage))
              (t                                    (my_reverse2 (cdr lst) (cons (car lst) storage)))
    )
)


"""----------------------------------------------------------------------------------------"""
"""-------------------                  Exercise 8.8                  -------------------"""
"""----------------------------------------------------------------------------------------"""

Write a procedure LAST-NTH that returns the nth element from the end of a given
list. Do NOT use NTH or ELT; use DOLIST.

2   '(6 5 4 3 2 1 0)        ->          2   (last 2 = first 4)    (length - 1) - n


(defun last-nth (n lst)

    (let    ((result            nil)
             (len               (- (- (length lst) 1) n))
             (counter           0)
            )
        (dolist (i lst result)

            (if     (= counter len)
                    (and (setf result (nth counter lst)) (setf counter (+ counter 1)))
                    (setf counter (+ counter 1))
            )
        )
    )
)

"""----------------------------------------------------------------------------------------"""
"""-------------------                   Exercise 8.9                   -------------------"""
"""----------------------------------------------------------------------------------------"""

See the PAIRLISTS in lecture notes. Define a procedure that “pairs” an arbitrary
number of lists. Here is a sample interaction:

* (pairlists '((a b) (= =) (1 2) (+ -) (3 9)))

                ((A = 1 + 3) (B = 2 - 9))


pairlist '( (a b) (= =) (1 2) (+ -) (3 9) )  nil nil)

(defun  pairlist  (x   list1  list2)


    (cond   ( (endp x)                  (cons (reverse list1)  (list (reverse list2))))
            ( t                         (pairlist  (cdr x)  (cons (caar x) list1)   (cons (cadar x) list2)))

    )
)

; (pairlist '( (a b) (= =) (1 2) (+ -) (3 9) )  nil nil)    will return     ((A = 1 + 3) (B = 2 - 9))


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

(defun pairlist2  (x)


    (append     (list (mapcar  #' car x))   (list (mapcar  #' cadr x)) )

)

"""----------------------------------------------------------------------------------------"""
"""-------------------                   Exercise 8.10                   -------------------"""
"""----------------------------------------------------------------------------------------"""

Define a procedure ENUMERATE that enumerates a list of items. Numeration starts
with 0. Define two versions, one with, and one without an accumulator.

CL-USER > ( enumerate '( A B C ))

        ((0 A ) (1 B ) (2 C ))




(defun range (x   storage)

    (cond   ( (eq x 0)              nil)
            ( (eq x 1)              (cons 0 storage) )
            ( t                     (range  (- x 1)  (cons (- x 1) storage) )  )
    )
)

(range 5 nil)

(0 1 2 3 4)


(defun enum_ (lst)

    (mapcar #' (lambda (x y)

        (list x y)
        )
    lst (range (length lst) nil))
)

"""----------------------------------------------------------------------------------------"""
"""-------------------                   Exercise 8.11                   -------------------"""
"""----------------------------------------------------------------------------------------"""


Write a program that takes a sequence, a start index, an end index and returns the
sub-sequence from start to (and including) end. Indices start from 0.


'(( a b c    d e f    g h )    3  5 )          ->      (d e f)


(defun sub-seq (lst start end )

    (let    ( (result                       nil)
            )
        (dolist  (i lst (reverse result))

            (if  (and (= start 0) (>= end 0))
                 (and (setf result (cons i result)) (setf end (- end 1)) )
                 (and (setf start (- start 1)) (setf end (- end 1)))
            )
        )
    )
)

"""----------------------------------------------------------------------------------------"""
"""-------------------                   Exercise 8.12                   -------------------"""
"""----------------------------------------------------------------------------------------"""

Given a sequence of 0s and 1s, return the number of 0s that are preceded by a 0.
Here is a sample interaction:

CL-USER > ( zeros '(1 0 0 0 1 0))

        2


(defun zeros (lst)

    (if     (not (= (car lst) 0) )
            (zeros (cdr lst))

            (let    ( (result               0)
                      (flag                 0)
                    )

                (dolist  (i lst (- result 1))

                    (if   (and (= i 0) (= flag 0))
                            (setf result (+ result 1))
                            (setf flag 1)
                    )
                )
            )
    )
)

"""----------------------------------------------------------------------------------------"""
"""-------------------                            END                   -------------------"""
"""----------------------------------------------------------------------------------------"""
```