; 2. Написать фукнцию которая получает как аргумент

; список чисел, а возвращает список квадратов этих чисел в том же порядке

(defun f2 (lst)

(mapcar #'(lambda (x)

(\* x x))

lst)

)

; (format t "~a~%" ( f2 '(1 2 3 4 5))) ; --> (1 4 9 16 25)

; 3. Напишите фукнцию, которая умножает на заданное

; число-аргумент все числа из заданного списка-аргумента, когда

; а) все элементы списка -- числа,

; б) элементы списка -- любый объекты.

; а) все элементы списка -- числа,

(defun f3a (lst num)

(cond ( (every #'numberp lst)

(mapcar #'(lambda (x) (\* x num)) lst))

)

)

; (format t "~a~%" ( f3a '(1 2 3 4 5) 2)) ; --> (2 4 6 8 10)

; (format t "~a~%" ( f3a '(1 2 3 4 5 d) 2)) ; --> NIL

; б) элементы списка -- любый объекты.

(defun f3b (lst num)

(mapcar #'(lambda (x)

(cond ((numberp x)

(\* x num))

(T x)))

lst)

)

; (format t "~a~%" ( f3b '(1 2 a b 5) 2)) ; --> ( 2 4 a b 10)

; (format t "~a~%" ( f3b '(a) 2)) ; --> (a)

; 4. Написать функцию, которая по своему списку-аргументу lst

; определяет является ли он палиндромом (то есть равны ли lst и (reverse lst)),

; для одноуровневого смешанного списка

(defun f4 (lst)

(every #'eql lst (reverse lst))

)

; (format t "~a~%" ( f4 '(1 a 3 a 1))) ; --> T

; (format t "~a~%" ( f4 '(1 2 3 1))) ; --> Nil

; 5. Используя функционалы, написать предикат set-equal, который возвращает t,

; если два его множества-аргумента (одноуровневые списки) содержат одни и те же

; элементы, порядок которых не имеет значения

(defun cmp\_set\_el (s el)

(some #'(lambda (x)

(eql x el)

) s )

)

; (format t "~a~%" ( cmp\_set\_el '(a 1 3 e) 'a)) ; --> T

; (format t "~a~%" ( cmp\_set\_el '(a 1 3 e) 10)) ; --> Nil

(defun set-equal (set1 set2)

(every #'(lambda (x)

(cmp\_set\_el set2 x)

) set1 )

)

; (format t "~a~%" ( set-equal '(a 1 3 e) '(e 1 a 3))) ; --> T

; (format t "~a~%" ( set-equal '(a 1 3 e) '(1 3 e))) ; --> Nil

; 6. Напишите функицю, select-select\_between, которая из списка-аргумента,

; содержащего только числа, выбирает только те, которые расположены

; между двумя указанными числами - границами-аргументами и возвращает их

; в виде списка (упорядоченного по возрастанию (+2 балла))

(defun between (x st end)

(cond ( (< st x)

(cond ( (< x end) T)

))

)

; (and (> st x) (< x end))

)

; (format t "~a~%" ( between 2 1 10)) ; --> T

; (format t "~a~%" ( between -2 1 10)) ; --> Nil

(defun select\_between (lstnum st end)

( sort

(remove-if-not #'(lambda (x)

(between x st end)

) lstnum)

#'<)

)

; (format t "~a~%" ( select\_between '(3 6 4 7) 1 10)) ; --> (3 4 6 7)

; (format t "~a~%" ( select\_between '(30 -6 4 7) 1 10)) ; --> (4 7)

; (format t "~a~%" ( select\_between '(30) 1 10)) ; --> Nil

; 7. Написать функцию, вычисляющую декартово произведение двух своих

; списков-аргументов (напомним, что A x B это множество всевозможных пар (a b),

; где a принадлежит A, b принадлежит B)

(defun pairs\_lst\_el (lst el)

(mapcar #'(lambda (x)

(list el x)

) lst)

)

; (format t "~a~%" ( pairs\_lst\_el '(30 2 3) 1)) ; --> ((1 30) (1 2) (1 3))

(defun decart (lsta lstb)

(apply #'append

(mapcar #'(lambda (x)

(pairs\_lst\_el lsta x)

) lstb))

)

; (format t "~a~%" ( decart '(1 2 3) '(4 5))) ; --> ((4 1) (4 2) (4 3) (5 1) (5 2) (5 3))

; 8. Почему так реализовано reduce, в чем причина?

; (reduce #'+ ()) -> 0

; (reduce #'\* ()) -> 1

; (format t "~a~%" (reduce #'+ ())) ; --> 0

; (format t "~a~%" (reduce #'\* ())) ; --> 1

; Грэм П. ANSI Common Lisp. – Пер. с англ. – СПб.: Символ-Плюс, 2012. – 448 с.

; (reduce function proseq функция &key key from-end start end initial-value)

; если initial-value не предоставлено, функция вызывается без аргументов

; В данном случае функции + и \* были вызваны без аругементов

; (format t "~a~%" (+)) ; --> 0

; (format t "~a~%" (\*)) ; --> 1

; Цитата из то же книги:

; (+ &rest ns) функция Возвращает сумму аргументов или 0, если аргументы не заданы.

; (\* &rest ns) функция Возвращает произведение двух аргументов или 1, если аргументы не заданы.

; 9. \* Пусть list-of-list список, состоящий из списков.

; Написать функцию, которая вычисляет сумму длин всех элементов list-of-list

; (количество атомов), т.е. например для аргумента ((1 2) (3 4)) -> 4.

; Вывод: Элементами list-of-list являются неструктурированные списки

(defun length\_lol (lol)

(reduce #'+ (mapcar #'length lol))

)

; (format t "~a~%" (length\_lol '((1) (3 4) () (a e r 4) ) ) ) ; --> 7