Задание 1

Условие: Напишите функцию которая уменьшает на 10 все числа из списка-аргумента этой функции, проходя по верхнему уровню списковых ячеек(* список смешанный, структурированный).

```
(defun map-minus-10 (lst)
    (mapcar #'(lambda (x) (- x 10)) lst))

(defun map-minus-10* (lst)
    (mapcar #'(lambda (x)
        (if (numberp x)
            (- x 10)
            x
        )) lst))


; Корректные запуски
(map-minus-10* '(1 2 3 4 5))
(map-minus-10* '(a 2 b 4 c))
(map-minus-10* '(1))
(map-minus-10* '(1 2 (3 4 5)))
(map-minus-10* '(a b))
(map-minus-10* '())

(map-minus-10 '(1 2 3 4 5))
(map-minus-10 '(1))
(map-minus-10 '())


; Некорректные запуски
(map-minus-10 '(a 2 b 4 c))
(map-minus-10 '(1 2 (3 4 5)))
(map-minus-10 '(a b))
```

Задание 2

Условие: Написать функцию, которая как аргумент получает список чисел, а возвращает список квадратов этих чисел в том же порядке.

```
(defun square (x) (* x x))
```

```
(defun squarelst (lst)
    (mapcar #'square lst)
)
```

Задание 3

Условие: Напишите функцию которая умножает на заданное число-аргумент все числа из заданного списка-аргумента, когда

1) все элементы списка – числа;
2) элементы списка – любые объекты.

```
; Для чисел
(defun mult (lst multiplier)
    (mapcar #'(lambda (x) (* x multiplier)) lst))

; Для любых объектов
(defun mult* (lst multiplier)
    (mapcar #'(lambda (x)
        (if (numberp x)
            (* x multiplier)
            x
        )) lst))
```

Задание 4

Условие: Написать функцию, которая по своему списку-аргументу lst определяет является ли он палиндромом (то есть равны ли lst (reverse lst)), для одноуровнего смешанного списка.

```
(defun palindromep (lst)
    (every #'eql lst (reverse lst))
)
```

Задание 5

Условие: Используя функционалы, написать предикат set-equal, который возвращает t, если два его множества-аргумента содержат одинаковые элементы, независимо от их порядка.

```
; Создаёт список длиной n из elem
(defun n-copy-list (n elem)
```

```
    (if (= n 1)
        (cons elem nil)
        (cons elem (n-copy-list (- n 1) elem))
    )
)

; Проверяет равны ли множества
(defun set-equal (set1 set2)
    (and
        (= (length set1) (length set2))
        (every #'member set1 (n-copy-list (length set2) set2))
        (every #'member set2 (n-copy-list (length set1) set1))
    )
)

(print (set-equal '(1 2 3 4 5) '(5 4 3 2 1)))
```

Задание 6

Условие: Напишите функцию 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)).

```
; Вставить число в отсортированный по возрастанию список
(defun insert-to-sorted (lst elem)
    (cond
        ((equal lst nil) (list elem))
        ((< elem (car lst)) (cons elem lst))
        (t (cons (car lst) (insert-to-sorted (cdr lst) elem)))
    )
)

; Соритирует список по возрастанию
(defun insert-sort (lst)
    (reduce #'insert-to-sorted (cons () lst))
)

; Предикат для проверки на то, что число между двумя числами.
; Если число находится между border1 и border2,
```

```
; возвращает его, иначе nil.
(defun betweenp (elem border1 border2)
    (if (or (and (<= elem border1) (>= elem border2))
        (and (>= elem border1) (<= elem border2)))
        elem
    )
)

; Возвращает список, в котором удалены nil элементы
(defun filter-nil (lst)
    (if (not (equal lst nil))
        (if (equal (car lst) nil)
            (filter-nil (cdr lst))
            (cons (car lst) (filter-nil (cdr lst)))
        )
    )
)

; Создаёт список длиной n из elem
(defun n-copy-list (n elem)
    (if (= n 1)
        (cons elem nil)
        (cons elem (n-copy-list (- n 1) elem))
    )
)


; Возвращает список из чисел, которые находятся
; между border1 и border2 включительно,
(defun list-beetwen (lst border1 border2)
    (filter-nil (mapcar #'betweenp
        lst
        (n-copy-list (length lst) border1)
        (n-copy-list (length lst) border2)))
)


; Возвращает отсортированный список чисел, которые находятся
; между border1 и border2 включительно.
```

```
(defun sorted-list-beetwen (lst border1 border2)
    (insert-sort (list-beetwen lst border1 border2)))
```

Задание 7

Условие: Написать функцию, вычисляющую декартово произведение двух своих списков-аргументов.

```
; Возвращает декаторо произведение двух списков
(defun decart (lst1 lst2)
    (mapcan #'(lambda (x)
        (mapcar #'(lambda (y)
            (list x y)
        ) lst2)
    ) lst1)
)
```

Задание 8

Условие: Почему так реализовано reduce, в чём причина?

— (reduce #'+ ()) -> 0

— (reduce #'* ()) -> 1

В случае, если reduce получает пустую последовательность аргументов, то она вызывает переданную функцию без аргументов и возвращает то, что вернёт она. Это сделано для того, чтобы переложить ответственность за значение по умолчанию на переданную функцию, так как на уровне reduce не известны особенности переданной функции и что она делает.

Задание 9

Условие: Пусть list-of-list список, состоящий из списков. Написать функцию, которая вычисляет сумму длин всех элементов list-of-list (количество атомов).

```
; Возвращает суммарную длину списков-элементов списка-аргумента
(defun list-of-lists (lst)
    (reduce #'(lambda (accum ilst)
        (+ accum (length ilst)))
    (cons 0 lst))
)
```