# Лекция 4 (09.03.2022)

\*(По умолчанию функции используют eql)

(**member** el lst) – возвращает список lst начиная с атома el  
(member '(a b) '(c (a b) d)) – если использует eql не обнаружит список (т.к. eql не сравнивает их)  
(member '(a b) '(c (a b) d) :test #'equal) – тогда будет обнаружен список

На прошлой лекции начали функционалы:

1. Применяющие (Например: **apply**, **funcall**)
2. Отображающие  
   (**mapcar** #'fun lst1) – проходит только по верхнему уровню элементов списка lst1  
   (**maplist** #'fun lst2) – применяет к хвосту, хвосту, хвосту…

mapcar, fun – должна быть одноаргументной, но если мы хотим несколько:  
(mapcar #'fun lst1 lst2 … lstk) но mapcar не проверяет длину списков lst и завершается когда кончается один из них.

Maplist, fun – тоже должна быть одноаргументной, но если мы хотим несколько, то также подаём несколько списков (maplist #'fun lst1 lst2 … lstk)

\*(**nconc -** ?переставляет указатели?)

**mapcan**, **mapcon** – объединяет результаты с помощью nconc

в mapcar, maplist – с помощью list

(**find-if** #'predicat lst) – возвращает первый элемент из списка lst который удовлетворяет predicat  
(**find-if** #'oddp '(2 4 7 5)) = 7  
(**find-if-not** #'predicat lst) – возвращает первый элемент из списка lst который **не** удовлетворяет predicat

(**remove-if** #'predicat lst)   
(remove-if #'oddp '(2 4 7 6 9)) = (2 4 6)

(**remove-if-not** #'predicat lst)  
(remove-if-not #'oddp '(2 4 7 6 9)) = (7 9)

\*(По умолчанию в функционалах функция всегда первый аргумент)

(**reduce** #'fun lst2) – применяет функцию fun каскадным образом к списку lst  
(**every** #'predicat lst) – T если каждый элемент удовлетворяет предикату  
(**some** #'predicat lst) – T если хотябы один элемент удовлетворяет предикату

Примеры:  
;1

**(defun** consist-of **(**lst**)**

**(if** **(member** **(car** lst**)(cdr** lst**))** 1 0**))**

**(defun** all-last-element **(**lst**)**

**(if** **(eql** **(**consist-of lst**)** 0**)(list** **(car** lst**))()))**

**(defun** collection-to-set **(**lst**)**

**(mapcon** #'all-last-element lst**))**

**(**collection-to-set **'(**i t i g t k s i f k**))**

;2

**(defun** decart **(**lstX lstY**)**

**(mapcan** #**'(lambda** **(**x**)**

**(mapcar** #**'(lambda** **(**y**)**

**(list** x y**))** lstY**))**

lstX**))**

**(**decart **'(**a b**)** **'(**1 2**))** **=** **((**a 1**)(**a 2**)(**b 1**)(**b 2**))**

\*(Знак # обязателен в лабораторных работах)

## Рекурсия

Классификация рекурсий:

1. Простая рекурсия – когда рекурсивный вызов встречается в теле функции один раз.
2. Рекурсия первого порядка – когда рекурсивный вызов встречается несколько раз.
3. Взаимная рекурсия – ?когда рекурсивные функцию вызывают друг друга.?

;рекурсия1

**(defun** my-member **(**el lst**)**

**(cond** **((null** lst**)** NIL**)**

**((equal** el **(car** lst**))** T**)**

**(**t **(**my-member el **(cdr** lst**)))))**

**(**my-member **'a** **'(**b a c**))** **=** T

**(**my-member **nil** **())** **=** **nil**

;поменяли проверки местами

**(defun** my-member **(**el lst**)**

**(cond** **((equal** el **(car** lst**))** T**)**

**((null** lst**)** NIL**)**

**(**t **(**my-member el **(cdr** lst**)))))**

**(**my-member **nil** **())** **=** T

;рекурсия2

**(defun** my-reverse **(**lst**)**

**(cond** **((null** lst**)** lst**)**

**(**T **(append** **(**my-reverse **(cdr** lst**))**

**(cons** **(car** lst**)** lst**)))))**

;еще один способ

**(defun** my-reverse1 **(**lst**)**

...**)**

**(defun** move-to **(**lst result**)**

**(cond** **((null** lst**)** result**)**

**(**T **(**move-to **(cdr** lst**)** **(cons** **(car** lst**)** result**)))))**

**(defun** my-reverse1 **(**lst**)**

**(**move-to lst **()))**