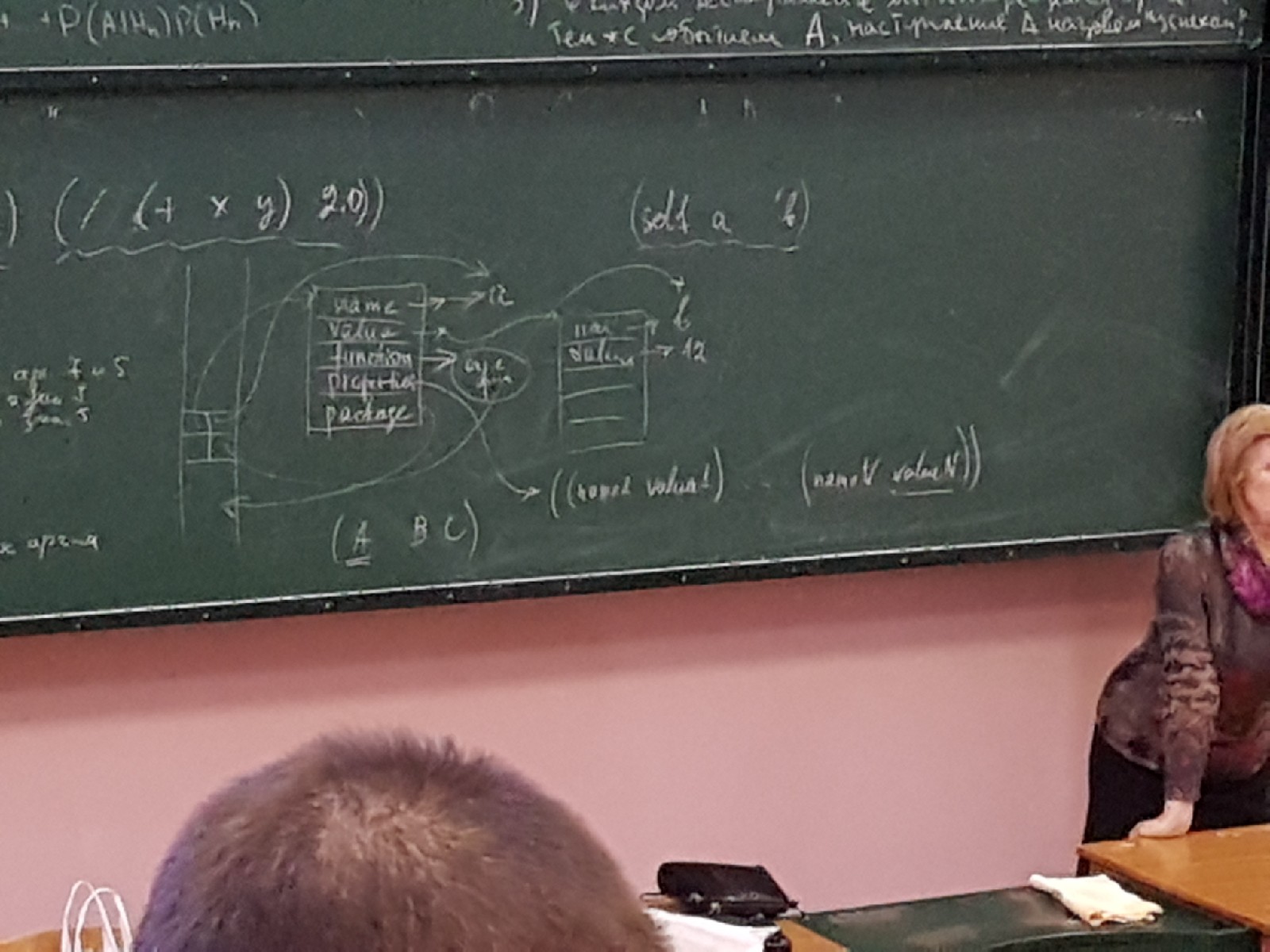
 (defun average( x, y ) ( / (+ x y) 2.0)

( average 7 5) //7 вычисляется к 7, 5 вычисляется к 5, вызов функции average с параметрами 7 и 5, создается переменная х со значением 7, создается переменная у со значением 5, после начинается обработка тела функции (/ (+ х у) 2.0). Запускается функция evel, запускается S-выражение. Запускается функция evel, попадает выражение (+ х у). Происходит анализ. Применяется функция + к аргументам, в результате возвращается 12. Второй аргумент вычисляется как 2.0, применяется функция деления к аргументам. Возвращается 6.0. В результате закрывается функция average

# Представление в памяти символьных атомов

Символьный атом в памяти представляется 5ю указателями:

* На имя- name-
* На значение- value-
* На функцию- function-
* На свойства- properties-
* На пакет- package-

В пакете хранится вся инфа о всех символьных атомах, которые уже существуют.  
Установить значение атома- (setf a ‘b).  
Символьный атом а может быть связан со значением, может быть с определением функции. В зависимости от контекста, может быть включён в формулу, вычислять её.  
Свойства принято в структурах. Свойства - это список из нескольких списков (( name1 value1)…(nameN valueN)), каждый их которых может быть символьным или определением функции. Это динамическая структура, которая хранит информацию об атомах, которые можно использовать в данной операции.

Атом в памяти хранится сложно (с).

Списки как частный случай выражения, точечная пара.

# Специальные функции

* Базисная функция cond  
  (cond (test1 body1)  
   (test2 body2)  
   …  
   (testN bodyN)  
   (T bodyT))
* (if test T.body F.body)
* (not S-выражение)
* (and arg1 arg2 … argN)- функция прорабатывает произвольное кол-во аргументов или не все.   
  And заканчивает работу на Null
* (or arg1 arg2 … argN)

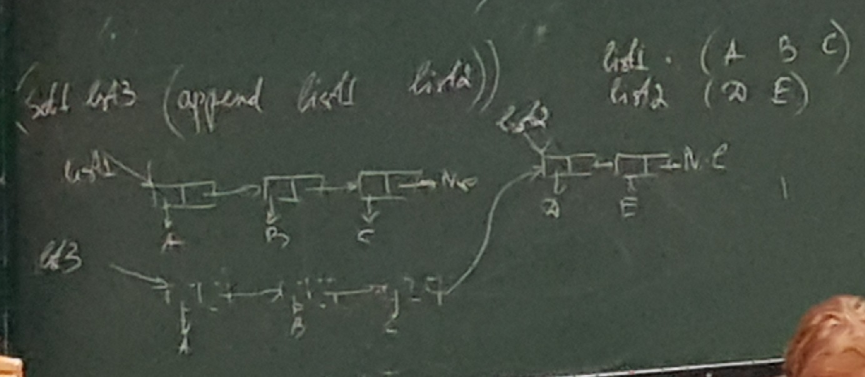
Будет сравнивать аргументы.  
(defun how\_alike (x y))  
 (cond ((or (= x y) (equal x y)) the\_same)  
 ((and (oddp x)(oddp y)) ‘both\_odd)  
 ((and (evenp x)(evenp y)) ‘both\_even)  
 (T ‘defference)))

У Let формы фиксированное кол-во аргумента  
(Let ((var1 value1)  
 (var2 value2)  
 (varN valueN))  
 body)

Let\* имеет жесткий отбор аргументов, медленнее работает, чем Let.

# Операции со списками

Функции можно разделить на 2 группы- структуро-разрушающие и неразрущающие структуры(работают медленнее, чем разрушающие).

* Функция append. Объединяет списки  
  (append list1 list2) list1 ( A B C ) list2 ( D E )  
  
* Функция nconc- n значит, что не создаются дубли
* Reverse обращает список только по верхнему уровню списка, работает с копией
* Nreverse- не делает копию, просто работает по верхнему уровню списка.
* Last- ищет последнюю списковую ячейку верхнего уровня
* Nth- возвращает голову N списковой ячейки, одноаргументная функция  
  (nth n spis)
* Nthcdr- возвращает остаток списка, начиная с n списковой ячейки  
  (nthcdr n spis)
* Length- Количество списковых ячеек верхнего уровня. Работает с любой структурой, которая представлена в виде списка
* Remove- удаляет  
  (remove el list)  
   (remove ‘el ‘(b a c)) -> (b c)  
   (remove ‘(a b) ‘((a b) c d)) -> ((a b) c d)  
   (remove ‘(a b) ‘((a b) c d); test #’equal) -> (c d)  
    
  (member el spis)- поиск элемента в списке
* (rplaca list el)- голова переустанавливается на el
* (rplacd list el)- переустанавливается указатель головы на el

//лучше уточнить пояснения функций, правильно ли услышала.