# Лекция 3 (02.03.2022)

## Повтор

Считать в Лиспе это, по сути, получение информации по адресу?

По поводу функций и их диаграмм: когда вызываем свою функцию, формальные параметры существуют только во время работы функции.

\*(В Лиспе принято называть “локальные переменные” лексическим атомом?)

Cond, if, and и or – функции формы

## Новая тема

Атому можно установить значение. Если это символьный атом, то память распределяется 5ти-указательным блоком. Есть локальные атомы и глобальные. Можно установить значение с помощью setf (не вычисляет первый аргумент, а второй аргумент вычисляет и устанавливается указатель value – может быть чем угодно (символ, список…)). Как только написали setf, это способ создать глобальный атом на всё время сеанса работы, пока мы не изменим это значение. Атом может быть одновременно и значением, и функциональным определением, и списком свойств.

## Начинаем знакомится с другими функциями

Есть такая функция let (является формой). Первый аргумент – это список.   
(let ((name1 value1)  
 (name2 value2)  
 ……………...  
 (nameN valueN))  
 body)

Let\* - работает более аккуратно, но медленней.

Связывание происходит в произвольном порядке – поэтому при определении следующего атома нельзя ссылаться на предыдущий. А в let\* можно сослаться, но она работает медленней.

## Функции использующиеся для работы со списками

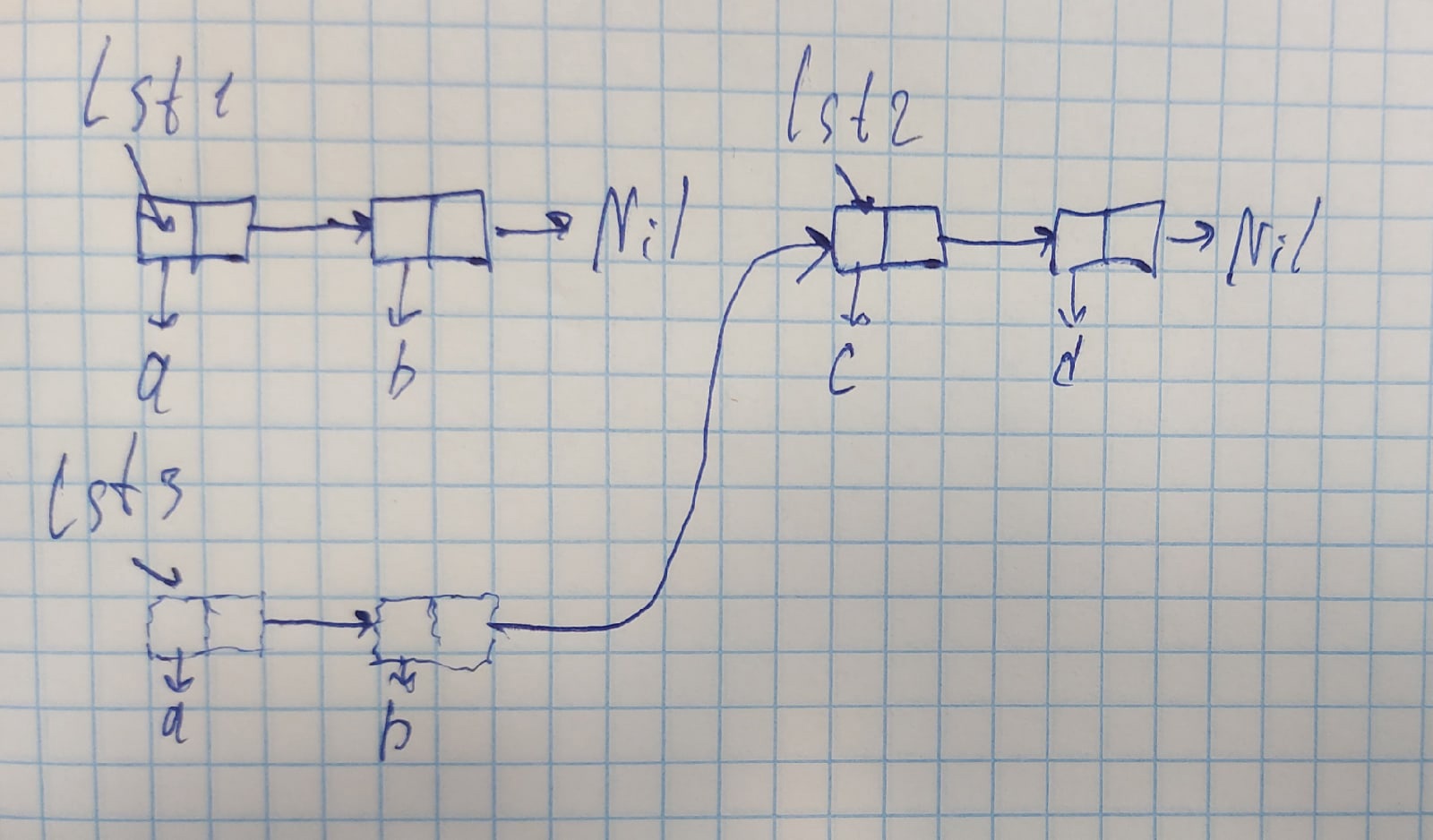
Их можно разбить на две группы:

1. Функции разрушающие структуры
2. Функции неразрушающие структуры

\*(Термин “среда” используется, когда загружается пакет – становиться доступен интерфейс этой среды. А что такое среда по отношению к памяти - непонятно)

\*(А что значит загружается Lisp?) (Что значит процесс и то, что он породился, если есть только адреса? (?создаётся адресное пространство?))

Нужно уметь создать, уничтожить, модифицировать и проверить список.

Функция **append** (соединяет списки). Функция append работает с копией (не разрушается структура).   
(setf lst1 ‘(a b))  
(setf lst2 ‘(c d))  
  
(setf lst3 (append lst1 lst2))

Создаёт копии всех аргументов кроме последнего. И последними указателями огни сцепляются. За счёт этого сохраняется возможность работы с lst1, lst2 и lst3. Но получается опасность, если поменять lst1, lst3 тоже изменится.

Функция **nconc** – ?объединяет списки "физически", устанавливая указатель в поле CDR последней ячейки списка, являющегося первым аргументом, на начало списка, являющегося вторым аргументом.?

Функция **reverse.**

Функция **nreverse** работает без копий.

Функция **last**.

(**nth** N lst) – N-ый элемент списка  
(**nthcdr** N lst) – список без N первых элементов  
(**remove** el lst) – принимает список (lst) и объект (el) и возвращает новый список, состоящий из тех же элементов, что и предыдущий, за исключением этого объекта.  
(**member** el lst) – проверяет есть ли элемент в списке  
**eql** – сравнивает атомы и числа (корректно сравнивает только числа одного типа) (часто используется во встроенных функциях)

Пока что разрешено использовать чужие функции, а потом запретят, например, append.

\*(Лисп старается избегать дублирования, поэтому надо стараться работать со множествами) (С одной стороны, зачем менять списки? (?лишние затраты?) С другой стороны, зачем создавать дубли? (?лишняя память?))

Ассоциативная таблица – это список из точечных пар. (Первый элемент воспринимается как ключ, второй как значение)

Но система не следит за уникальностью ключей.

Итак: есть возможность работать со множествами и с ассоциативными таблицами.

## Функционалы

\*(Различают три основных **типа** **алгоритмов**: линейные, разветвленные, циклические.)

Для организации повторных вычислений используются либо функционалы, либо рекурсия.

**Функционалы**. (Функции которые в качестве аргумента принимают функции или возвращают функции в качестве результата) (Если в качестве результата возвращается функция, говорят функция более высокого порядка)

Функционалы можно разделить на две группы:

1. Применяющие (Например: apply, funcall)  
   (**apply** #’fun lst) – нужно применить fun к lst  
   #’ – функциональная блокировка, то же самое что и function (фиксирует значение функции, когда она начинает работать)  
   fun – либо имя, либо лмбда-выражение  
   lst – список аргументов  
   (**funcall** #’fun arg1 … argN)
2. Отображающие (Более употребимые) (Функционалы, позволяющие реализовать повторные вычисления) (Многократное применение функции формирует список результатов)  
   (Наиболее частые: mapcar и maplist)  
   (**mapcar** #’fun lst1) – функция (fun)(м.б. именем или лямбда-выражением) применяется к каждому элементу списка (lst1), на выходе получается список из этих результатов. (fun желательно должна работать и с атомами, и со списками)  
   (**maplist** #’fun lst2) – многократно применяет функцию (fun) с списку (lst2), а потом к хвосту, пока не хвост не будет пустым списком.