

# Голливудское начало Борисовна

07.02.23. 1/р. 1

- [•] Лицо (физич. прог-е) - смотрят, пока проверяют его, поглазят.

• Красота на фотографии + рабочий

• Справа от а/р - надо принять из а/р

• Ответ: красота не важна  
(хотя можно накинуть пухл + румянами. ответ.)

"На камере а/р начинает со спр. - установка опроса

Методология прог-я - осн. принципы, отк. ответ си. прог.  
от других (виды прог-я)

1) инструктивная

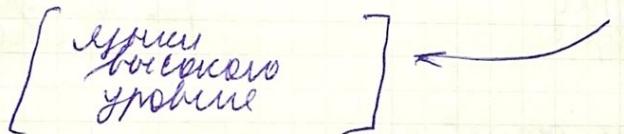
• Исправ. реакции (из команду)

• инструкции - инс. действие

• математической когн. (помощь в отработке умения)

асемблер (симв. язык) - интенсивность

программ  
(т.к. уверен.)



Программа - набор команд. - структур. текст, ест.  
из операторов (команд)

команда → оператор

данные → оператор

конкр. исп. действ. → оператор

- Примечание! команда → набор инструкций  
(перевод)

Символическое описание

Объект - набор данных и методов,

↑ способ обработки данных и параметров для  
работы с ними (способ структуризации)

ОПГ - не другие методологии, а другой вид логики;  
это все равно в чём - одна-же система операторов.

Румянцевское программирование

Физике - отображение обращение ии-ва значений арг. в  
ии-во значенией физики.



## Lisp - List Processing

Числа -

- список надо пустой, надо начинать

ЯП.

бесконечный      иначай      единичное  
таблица            таблица            - Lisp.  
(аргумент - показание)

Тип -

Описания переменных: имена, они  
зарегистрированы в компиляторе, чтобы было удобно,  
когда будет компилироваться.

кем нечет,  
кому?

Причина боязни при обработке описаний переменной

• расположение в списке пакета - не ясно.

Система имен (таблица имен):

имя - адрес  
имя - адрес  
...

→ быстрое  
выводим?

Удалить - пока не станет пустым

Описка - имена  $\Rightarrow$  в строке  $\Rightarrow$  удалить из списка

Обработка Lisp:

Символическая обработка

С т.п. нужно писать программы и данные - списки.

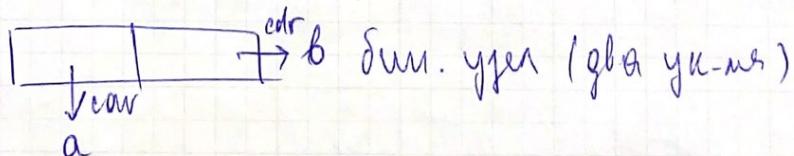
$\Rightarrow$  программа и ее воспринимает себе как данные  
и выполняет, данные работают

Атомы: имена символов без пробелов

Конструкции (,) - признак структ.

Горизонтальная - разделяет (а. б.)

Тип - первое значение



адр - content of decrement ("каз зап")  
вар - content of address register ("указ")

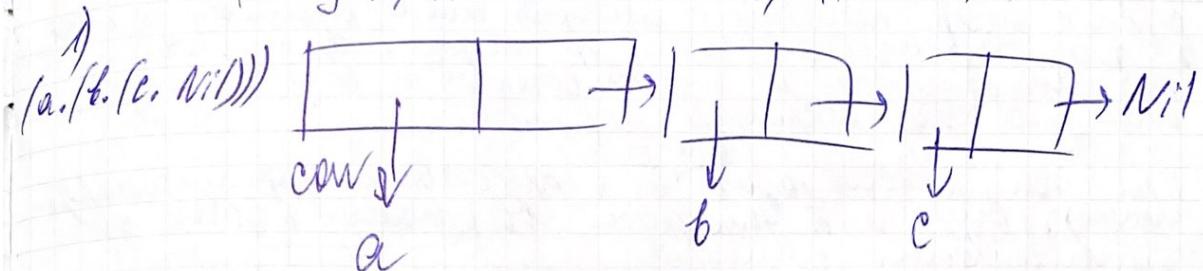
S-блокчукъ := (сущн) | (именное имя).

Синтакс - модель S-блока

Синтакс := (выраж) | (вычисл.)

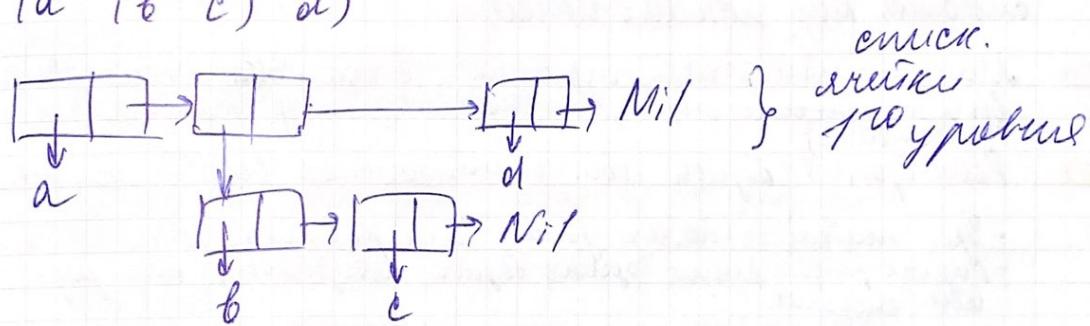
(выраж) := () | Nil,

(вычисл) := (<знача>, <хвост>)

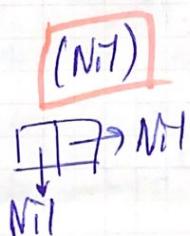


об обр. дерево:

i) (a (b c) d)



аналогич - на лекции



## 14.02.24. Лекция 1.

Ручное - базовое понятие.

Lisp.

- 1958 г. - предложен,
- поддерживает метод ФП;
- анализативный.

Методология ФП опр. процесса восчисления, на основе строих обстр. понятий и методов симв. обработки данных, а также допускает восемь ур. агрегации базовых восчислений для каждого типа объектов и опр. различную систему символов, отражающую их природу.

При этом классы язар и их представление определяются формулами языка, т.е. указанием новых функциональных символов.

В программах: кусок программы, переменные.

Опн. Система в ког. допускается включение символов с опр. смыслом, наз. анализативной.

Опн. Минималистичный набор обозначений, к которому можно свести все восчислительные формулы системы, наз. базисом.

Опн. Реализующие базиса яви. минимальной версии системы.

- При построении новых язар базис расширяется.
- Раннее расширение одного языка могут вовлк. как разные системы.

### Математические основы ФП.

Цели выделения языка Маркаро (1956-1960).  
Она основана на исчислении Тирса (2-исчисление).

- все действия организованы в виде функций.

Оси. на восчислении двух тип. симв. и свободных переменных?

Символика обработка в Lisp реализуется путем интерпретации S-выражений, состоящих из символов и аргументов, аргументы которых упорядочены.

#### Классификации функций

##### ① Чистые функции

- строго математические; неизменяющие
- символы воен-се аргументов, затем - значение функции
- возвращают чистой результат:
  - он не зависит от свободных переменных (подавляемых переменных)

##### ② Смешанные функции (программы)

- не строг. образуют обраб. аргументов (меняют не вовл.)

##### ③ Несвободные функции

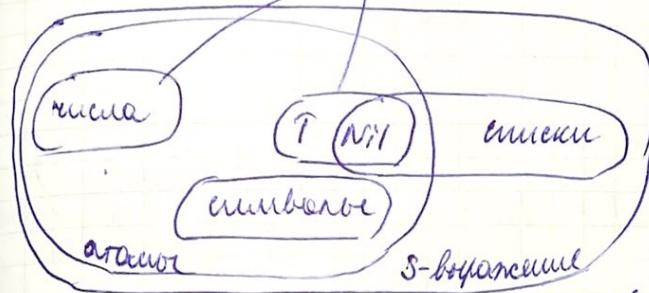
- связывают оп. опор. на устройство (первичн. ког. языке)

(4) Ф-ии высших порядков (Более высоких) (структурированы)

- либо в качестве аргумента получают ф-ио, либо возвращают ф-ио в качестве параметра;
- удобны для синтаксически упр-шего констр.-я программы.

инвертирование линейное

демонстрирует самовоспоминание



Базисы языка

- атомы
- символические узлы (для представления структур)
- базовое функции, функциональное:

	atom	
car	car	- обозначивает переход по CAR указателю
кадр	cadr	- обесц. "CDR" -
	eq	- сравнение (второй Т или NY)
комбинация	quote	- обозначение конструкции как данных.
	cond	
	eval	
	function	
	cons	

? Имена написаны на Схеме.  
Имена консервативно изменяются

Маг базисные выражения функций:  $(\lambda \text{ var } \text{ arg}_1 \text{ arg}_2 \dots \text{ arg}_N)$

Пример:  $(+ 1 2)$  аниски  
 $(\lambda (\text{arg} \text{ arg}_1 \text{ arg}_2 \dots \text{ arg}_N) \text{ cons} \text{ quote})$  1<sup>м</sup> арг.-объектально ищет арги  
 $\underbrace{(+ 1 3)}_{\text{справа воспринимается арг-ом}} (\lambda (\text{arg} \text{ arg}_1 \text{ arg}_2 \dots \text{ arg}_N) \text{ cons} \text{ quote})$

Структурированные списки; предикаты итерации.

По умолчанию система воспр. любую конструкцию как программу.  
Но это не так, потому воспринимать ее как программу - quote:

$(\text{quote} (A B C)) \sim ((A B C))$  - краткая форма  
 вырова ф-и quote

Проверка восприятия

Можно блокировать атомы.

Блокировать самовоспоминание атомов не нужно.

Имена, Т, NY - самовоспоминание (аргумент) - это  
именно само имена...  
Вспомните "воспоминать"  
etc (единаково)  
получить  
здесь же  
в конц. напи.

След. арги - после 3-х итераций  
таких нет.

## Маг. основы языка или λ-выражение.

В соотв. с именем. Первая логика оп-а языка. Ключ:

$\lambda(x_1, x_2, \dots, x_n). f_n$  - λ-выражение

λ-аргументы аргументы ф-ция  
аргументов → ф-ция  
λ-аргументов

Пример. {  $\lambda(x). x + 1$

$\lambda(x). x + 1 + y$  ← не чистая оп-а  
← внешнее месо выражения

$x_1, x_2, \dots, x_n$  - опред. параметров

(lambda  $(x_1 x_2 \dots x_N) f_n$ ) - (λ-выражение)  
λ-аргументы оп-и без имени  
λ-аргументы → s-выражение

Пример. { (lambda (x y) (+ (\* x x) (\* y y)))

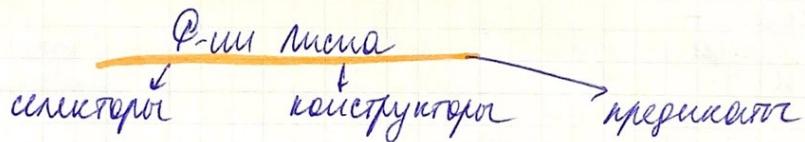
Как воспользоваться λ-функцией без имени:

(λ-выражение  $a_1 a_2 \dots a_N$ )

практические напр.

Defn. defn - базис наиболее часто используемое оп-и.  
(употребительное)

(defun (имя) (список аргументов) тело )



### 1) селектор

- car - общи. переход по CAR-ук. и CDR-ук. к концу пачки языка,
- cdr - переход по CDR-ук. к концу пачки языка,

допуск. к-во аргументов(), входит лишь применение только к структурам (не к атомам) [допуск в Lisp-расширении]

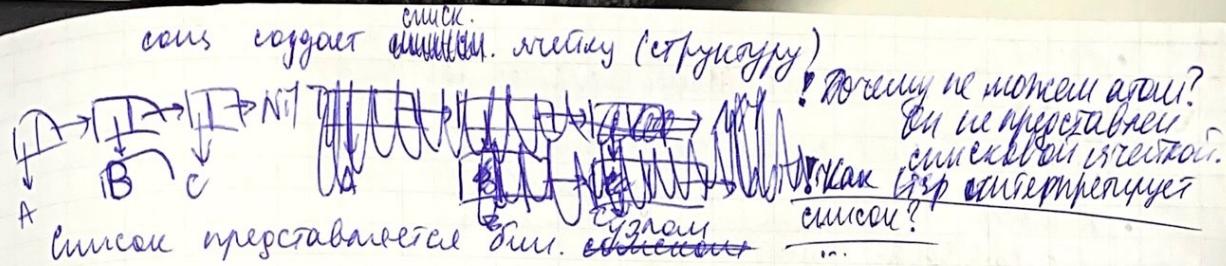
2) В математике скобки - признак структур.

### 2) конструктор

- cons - матем. двухкомпонентной  
одновременно связывающей структуры ~~и~~ расшир. матем.

(cons arg1 arg2)

Пример. { (cons 'A 'B C))



cons 'A' 'B) A . B  $\xrightarrow{\downarrow} B$  ! cons есть либо список,  
либо пор. пара

- (не падает) list - группа;  
проверяется число арг. б.;  
всегда возвращается список.

Пример } (list 'A 'B 'C)

! list реализован с помощью cons.  $\Rightarrow$  на падах неоднозначно.  
реализация списка с помощью cons

### 3) установка

19.02.24. Лекция 2.

#### 3) функции установки (prefixes)

- нужны для того, чтобы устанавливать структуру ф-ко.

! Для Lisp всегда возвращаются значения.

Рим - защищает (иссет защищенный х-р)  $\Rightarrow$  появление префиксов.

##### • atom

- один аргумент

- проверяет: атом, не атом?

##### • consp

- пор. пара, не пор. пара

##### • listp

- список, не список

{ Рим - защищает пару, второй из которой - список / хвост, кот. также  
необходимо проверять на то, является ли он списком.

Следует списка и пор. пары.

##### • Null

- пустой список / пустой список?

##### • eq

- применение только к симв. атомам, сравнение - по значению.

##### • eql

- основание к численные

- применение к символам и числам (оригинал представления)  $\neq$  по значению.

##### • =

- применение только к числам по значению

- equal
  - сравнение списков (аргв., число - одинак. предс.)
- equalp
  - сравнение списков, чисел, символов  
но не строк
- numberp
  - проверка число/не число?
- symbolp
  - проверка символ/не символ?
- zero?
  - проверка на 0
- < >
  - проверка порядка
- oddp, evenp
  - проверка четн.

Пример: member содержит eval:

{ (member '(a b) '(c (a b) e)) → NIL  
т.к. eval не умеет "уравнивать" числа.

То есть? Если ~~функция~~ делает все "карециенно" - приведеет список к именам символов.

Возможные функции.

Возможные выражения.

Система по умолчанию воспринимает первое значение списка как цель функции, а остальные - как аргументы функции.

[ впр - отнести ошибку исправить и]

? Помогите определение - рекурсивное определение функции.  
(помогает снизить способ реализации)

#### • eval

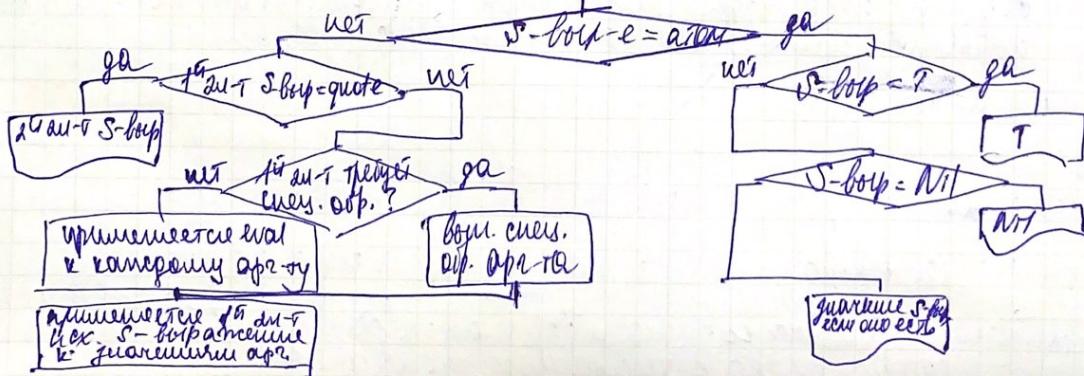
- запускается автоматически при наборе s-выражения;
- грамматика - интерпретируется;
- всегда выдаёт последнее полученное значение

Если работает lisp - работает со списками, указателями. (но не аргументами)  
⇒ print меняет, т.к. lisp и так выдаёт отпечаток.

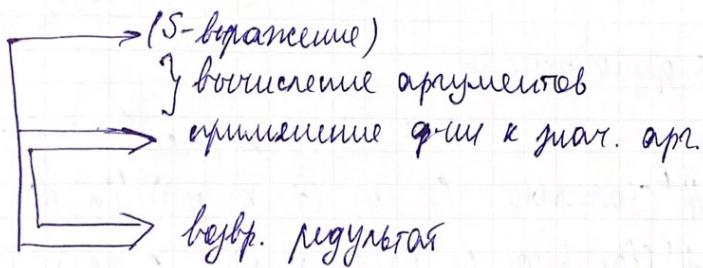
Логика работы

Методика вычисления  
исправления

[eval s-выражение]

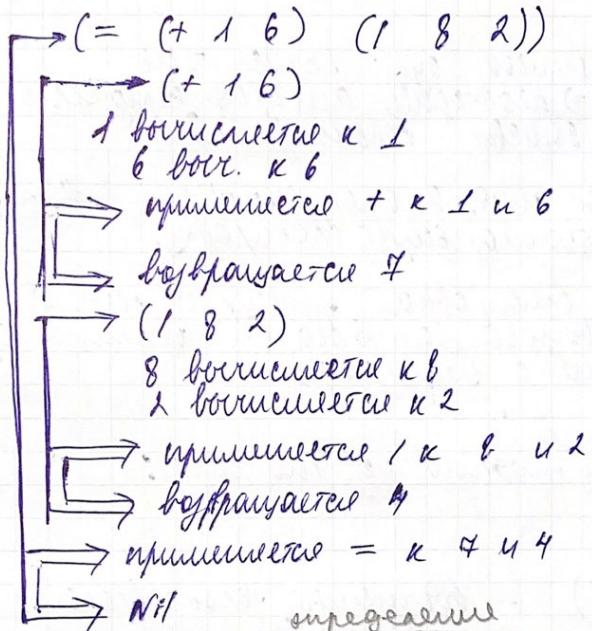


## Доказывание вложимости



Пример

①



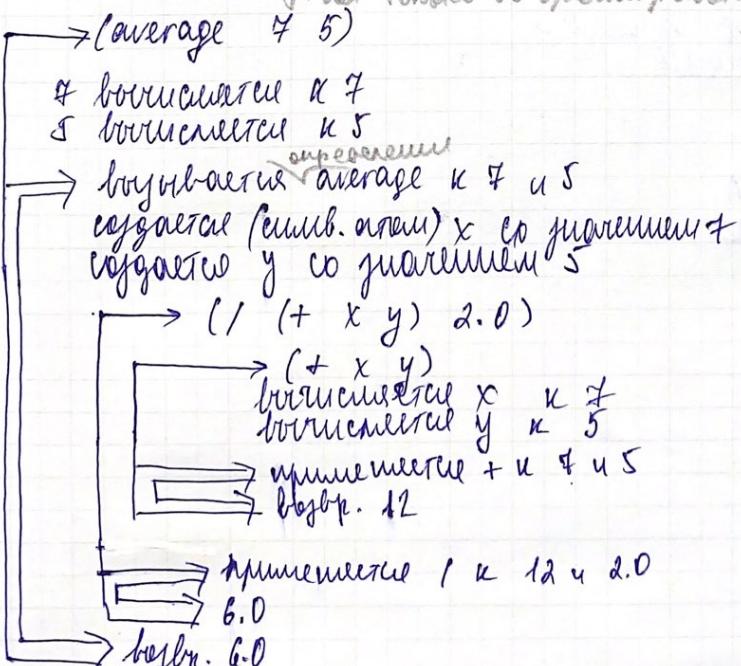
②  $(\text{defun average } (x y) (/ (+ x y) 2.0))$  defun - опр. ф-и

макрос?

лексик.

использование (перем.)

7-ия только во время работы ф-ии



3 функциональное выражение где больше lambda-выражений  
(суперпозиция)

- apply
- funcall

один из них для  
исчисления

(применение функционального)

Пример Определение выражения с иер. apply и funcall:

(apply #'(lambda (x y) (+ x y)) '2 5))

(funcall #'(lambda (x y) (+ x y)) 2 5)

можно использовать  
функцию apply

Проблема apply: список будет возвращаться  
→ необходимо использовать ~~перед списком~~  
список - только значение.

#' - первый символ списка, идет, аналог function #'<sup>i</sup>/function  
Она ведет функциональную блокировку.

Заречи? Ручная блокировка приводит к созданию памяти  
на момент выполнения lambda (и список будет возвращаться  
тому кто запросил с этим состоянием).

I Как работает с атомами?

Символический атом можно установить в ядре:

• setf

(setf A 12) - возвращает только адрес  
других

• set

(set 'A 12) - возвращает значение аргумента

представление в памяти  
символического атома.

δ упоминается

name	→ A
value	→ 12
function	→
properties	→
package	→