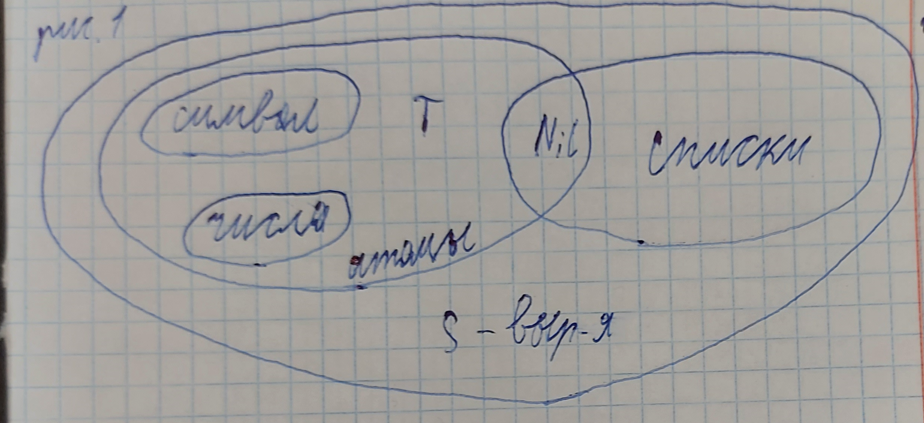
# Лекция 2 (16.02.2022)

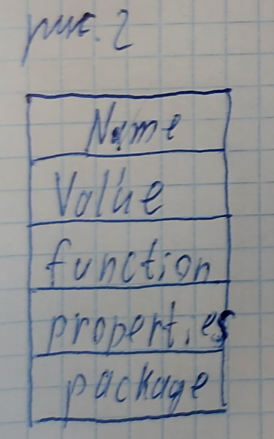
LISP – язык символьной обработки. Конструкции ЛИСПА синтаксически минимальные. В языке программирования можно использовать понятие базиса (…).

(Рисунок 1)



Атом представляется пятью указателями

(Рисунок 2)



Всю информацию, которую обрабатывает компьютер это программа и данные. В Лиспе по умолчанию всё воспринимается по умолчанию как программа.

Классификация функций. (классифицировать модно множеством способов, приведём один из них)

* Чистые функции. (“Чистые” математические функции, как в математике)  
  (Без побочных эффектов)  
  \*(результаты функций в Лиспе это S-выражения)
* Рекурсивные функции.
* Специальные функции или формы.  
  Отличаются от чистых функций тем, что у них может быть произвольное (переменное) кол-во аргументов (вчера 1, а завтра 25) или они обрабатываются по-разному. (Например: базисная функция cond; в функции if 3ий аргумент можно не указывать (будет nil); and и or)
* Псевдофункции. (создают спецэффекты (побочные эффекты), например на экране)
* Функции с вариантами значений.
* Функционалы / функции более высокого порядка. (функции которые в качестве аргумента принимают функции или возвращают функции в качестве результата)  
  (позволяют создавать синтаксически управляемые программы)

Другой вариант классификации:

* Базисные функции (Базис Лиспа – базисные синтакс. конструкции и базисные функции – минимальный набор функций, позволяющих решить любые задачи (car, cdr, cons, *определить природу аргумента, атом это или бинарный узел*) + минимальный набор форм (cond); функционалов (apply, funcall); еще несколько спец. функций (quote - ‘); eval (evaluate – вычисли, запускается без нашего участия. Является интерпретатором.) и lambda (базовый способ создания функций без имени – лямбда выражения))  
  \*(Лисп написан на Лиспе)   
  \*(Лисп – это обработчик списков)   
  \*(Eval – это интерпретатор, на вход ему может прийти только S-выражение. Вычислить это перейти по указателю значения.)  
  \*(lambda – определение локальной функции)  
  \*(Лисп первый столкнулся с проблемой с памятью, первый создал сборщик мусора)
* …?? И все остальные

## Способ определения функций.

\*(Лисп — это расширяемый язык / расширяемая система / консервативно расширяемый (не меняет базис))

\*(Программист думает только о цели (не нужно думать, например, о памяти))

### Лямбда определение функций

Лямбда выражения — это список. Первый элемент это lambda. Потом лямбда-список, а затем тело функции или форма.

лямбда-выражение = (lambda лямбда-список форма)

лямбда-список = (лямбда-выражение последовательность форм)

### Функции с именем

(defun f лямбда-выражение)

(defun f(x1, …, xk) (форма))

\*(лучше использовать лямбда определение – выгодней (по памяти скорости?))

### Специальная функция let

Аналог лямбда-определения. ?Только связывание … параметров вынесено в начало?

(let (x1 p1) (x2 p2) … (xk pk) e) = ((lambda (x1 x2 … xk) e) p1 p2 … p3) = лямбда-определение

### ?Функции?

1-ая группа селекторы: car и cdr – чистые функции (работают только со структурами)

\*(В Лиспе всегда возвращается последнее вычисленное значение)

2-я группа конструкторы: cons и list

Функция cons – предназначена для того, чтобы создавать список, создаёт списковую ячейку. На вход всегда 2 аргумента (s-выражения)

Функция list (не базисная) – всегда возвращает список, является формой (получает произвольное кол-во аргументов)

3-я группа предикаты:

\*(Всё что не Nil, то True) \*(Спец. символ T тоже True)

Функция Null – применима к структурам, пустая или непустая?

Функция listp – список или не список (символ p в конце означает предикат)

Функция consp – представлена ли структура списковой ячейкой

\*(всё это одноаргументные функции)

\*(В Лиспе не дублируются данные)

\*(Есть понятие сеанс работы – во время него можно расширять ядро…)

Функция eq – это тоже самое или нет? (equal) - чистая базисная функция с двумя аргументами, работает с символьными атомами (сравнивает указатели на них). Не сравнивает числа.

Функция eql – сравнивает атомы и числа (корректно сравнивает только числа одного типа) (часто используется во встроенных функциях)

Функция = – применима только к числам (разных типов корректно)

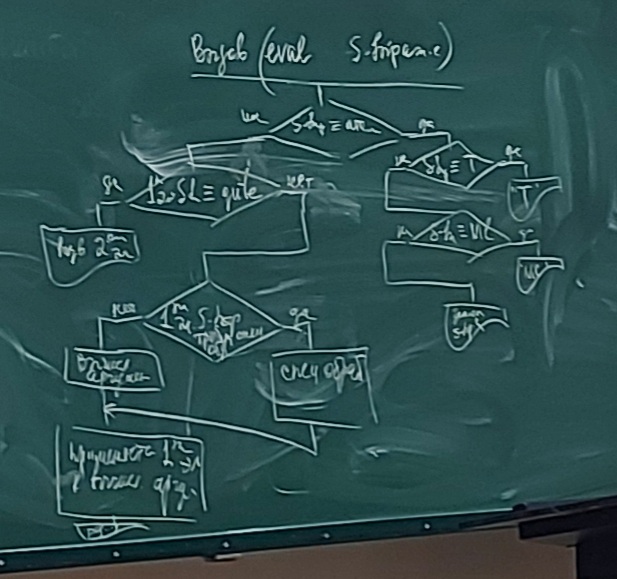
Функция equal – сравнивает атомы и списки, но не сравнивает числа

Функция equalp – сравнивает всё

### Заголовок?

\*(Если атом стоит первым в списке, то система по умолчанию работает с указателем function)

\*(Eval запускается автоматически, то есть по умолчанию)

Вызов (eval S-выражение)  


Система должна провести синтаксический разбор:

1. S-выражение атом или нет?
2. Если атом => S-выражение == True? Если да, то появляется T на экране  
   если nil, то печатает nil,  
   если нет, то возвращает значение S-выражения,  
   если нет значения, то ошибка.
3. Если не атом => S-выражение является квотой? (quote ‘)  
   если да, то возвращается второй элемент

Если не квота, то это функция => первый элемент S-выражения требует спец. обработки?

1. Если да, то выполняется спец. обработка,  
   если нет, то вычисляются аргументы с помощью eval и после этого применяется первый элемент (функция) к вычисленным аргументам.
2. Возвращается результат (результат это S-выражение)