



Министерство науки и высшего образования Российской  
Федерации  
Федеральное государственное бюджетное образовательное  
учреждение высшего образования  
«Московский государственный технический университет  
имени Н.Э. Баумана  
(национальный исследовательский университет)»  
(МГТУ им. Н.Э. Баумана)

---

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчет по лабораторной работе №2  
по дисциплине «Функциональное и логическое  
программирование»

Тема Определение функций пользователя

Студент Зайцева А. А.

Группа ИУ7-62Б

Оценка (баллы) \_\_\_\_\_

Преподаватели Толпинская Н.Б., Строганов Ю. В.

Москва — 2022 г.

# Теоретические вопросы

## 1. Базис Lisp

Базис – это минимальный набор инструментов языка и структур данных, который позволяет решить любые задачи.

Базис Lisp :

- атомы и структуры (представляющиеся бинарными узлами);
- базовые (несколько) функций, функционалов и форм: встроенные — примитивные функции (atom, eq, cons, car, cdr), которые носят частичный характер; формы (quote, cond, lambda, eval); функционалы (apply, funcall).

Атомы:

- символы (идентификаторы) – синтаксически – набор литер (букв и цифр), начинающихся с буквы;
- специальные символы – T, Nil (используются для обозначения логических констант);
- самоопределимые атомы – натуральные числа, дробные числа, вещественные числа, строки – последовательность символов, заключенных в двойные апострофы (например, “abc”);

Более сложные данные – списки и точечные пары (структуры), которые строятся с помощью унифицированных структур – блоков памяти – бинарных узлов.

Определения:

Точечная пара ::= (<атом> . <атом>) | (<атом> . <точечная пара>) | (<точечная пара> . <атом>) | (<точечная пара> . <точечная пара>);

Список ::= <пустой список> | <непустой список>, где

<пустой список> ::= () | Nil,

<непустой список> ::= (<первый элемент> . <хвост>),

<первый элемент> ::= <S-выражение>,

S-выражение ::= <атом> | <точечная пара>,

<хвост> ::= <список>.

Функцией называется правило, по которому каждому значению одного или нескольких аргументов ставится в соответствие конкретное значение результата.

Функции всюду определены (то есть результат есть всегда), их аргументы и результаты – S-выражения.

Функционалом, или функцией высшего порядка называется функция, аргументом или результатом которой является другая функция.

Форма – функция, которая особым образом обрабатывает свои аргументы, т. е. требует специальной обработки.

## 2.Классификация функций

1. Чистые математические функции (имеют фиксированное количество аргументов, сначала выясняются все аргументы, а только потом к ним применяется функция);
2. Рекурсивные функции (основной способ выполнения повторных вычислений);
3. Специальные функции, или формы (могут принимать произвольное количество аргументов, или аргументы могут обрабатываться по-разному);
4. Псевдофункции (создают «эффект», например, вывод на экран);
5. Функции с вариантами значений, из которых выбирается одно;
6. Функции высших порядков, или функционалы – функции, аргументом или результатом которых является другая функция (используются для построения синтаксически управляемых программ);

Классификация базисных функций и функций ядра.

1. Селекторы: `car` и `cdr` (будут подробнее рассмотрены ниже).
2. Конструкторы: `cons` и `list` (будут подробнее рассмотрены ниже).
3. Предикаты – «логические» функции, позволяющие определить структуру элемента:
  - `atom` возвращает `T`, если значением её единственного аргумента является атом, иначе – `NIL`;
  - `null` возвращает `T`, если значение его аргумента – `NIL` (пустой список), иначе – `NIL`; `listp` возвращает `T`, если значением её аргумента является список, иначе – `NIL`; `conspr` возвращает `T`, если значением её аргумента является структура, представленная в виде списковой ячейки, иначе – `NIL`.

#### 4. Функции сравнения (принимают два аргумента, перечислены по мере роста «тщательности» проверки):

- `eq` корректно сравнивает два символьных атома. Так как атомы не дублируются для данного сеанса работы, то фактически сравниваются соответствующие указатели. Возвращает `T`, когда: 1) значением одного из аргументов является атом, и одновременно 2) значения аргументов равны (идентичны). В ином случае значением функции `eq` является `NIL`.
- `eq1` корректно сравнивает атомы и числа одинакового типа (синтетической формы записи). Например, `(eq1 1 1)` вернет `T`, а `(eq1 1 1.0)` – `Nil`, так как целое значение `1` и значение с плавающей точкой `1.0` являются представителями различных классов;
- `=` корректно сравнивает только числа, причем числа могут быть разных типов. Например, `(= 1 1)`, и `(= 1 1.0)` вернет `T`;
- `equal` работает идентично `eq1`, но в дополнение умеет корректно сравнивать списки (считая списки эквивалентными, если они рекурсивно, согласно тому же `equal`, имеют одинаковую структуру и содержимое; считая строки эквивалентными, если они содержат одинаковые знаки);
- `equalp` корректно сравнивает любые `S`-выражения.

### 3. Способы создания функций

Определение функций пользователя в Lisp-е возможно двумя способами.

- Базисный способ определения функции - использование  $\lambda$ -выражения ( $\lambda$ -нотации). Так создаются функции без имени.

$\lambda$ -выражение: `(lambda  $\lambda$ -список форма)`, где  $\lambda$ -список – это формальные параметры функции (список аргументов), а форма – это тело функции.

Вызов такой функции осуществляется следующим способом: `( $\lambda$ -выражение последовательность_форм)`, где последовательность\_форм – это фактические параметры.

Вычисление функций без имени может быть выполнено с использованием функционала `apply`: `(apply  $\lambda$ -выражение последовательность_форм)`.

Функционал `apply` является обычной функцией с двумя вычисляемыми аргументами, обращение к ней имеет вид: `(apply F L)`, где `F` – функциональный аргумент и `L` – список, рассматриваемый как список фактических па-

раметров для F. Значение функционала – результат применения F к этим фактическим параметрам.

- Другой способ определения функции – использование макро-определения `defun`:

(`defun` имя\_функции  $\lambda$ -выражение),

или в облегченной форме:

(`defun` имя\_функции  $(x_1, x_2, \dots, x_k)$  (форма)), где  $(x_1, x_2, \dots, x_k)$  – это список аргументов.

В качестве имени функции выступает символьный атом. Вызов именованной функции осуществляется следующим образом: (имя\_функции последовательность\_форм), где последовательность\_форм – это фактические параметры

$\lambda$ -определение более эффективно, особенно при повторных вычислениях.

Параметры функции, переданные при вызове, будут связаны с переменными в списке параметров из объявления функции. Еще один способ связывания формальных параметров с фактическими – использование функции `let`:

(`let` (( $x_1$   $p_1$ ) ( $x_2$   $p_2$ ) ... ( $x_k$   $p_k$ ))  $e$ ),

где  $x_i$  – формальные параметры,  $p_i$  – фактические параметры (могут быть формами),  $e$  – формам (что делать).

#### 4. Функции `Car` и `Cdr`

Функции `car` и `cdr` переходят по соответствующему указателю аргумента (бинарного узла).

Функция `car` от одного аргумента возвращает первый элемент списка, являющегося значением её аргумента.

Функция `cdr` возвращает хвост списка, являющегося значением её единственного аргумента (хвостом, или остатком списка является список без своего первого элемента).

Современные диалекты Лиспа обычно допускают для функций `car` и `cdr` мнемоничные синонимичные названия: `first` и `rest` соответственно.

#### 5. Назначение и отличие в работе `Cons` и `List`

`CONS` принимает 2 указателя на любой S-выражения и возвращает новую `cons`-ячейку (списковую ячейку), содержащую 2 значения. Если второе значение

не NIL и не другая cons-ячейка, то ячейка печатается как два значения в скобках, разделённые точкой (так называемая точечная пара). Иначе, по сути, эта функция включает значение первого аргумента в начало списка, являющегося значением второго аргумента.

Функция list, составляющая список из значений своих аргументов (у которого голова – это первый аргумент, хвост – все остальные аргументы), создает столько списковых ячеек, сколько аргументов ей было передано. Эта функция относится к особым, поскольку у неё может быть произвольное число аргументов, но при этом все аргументы вычисляются

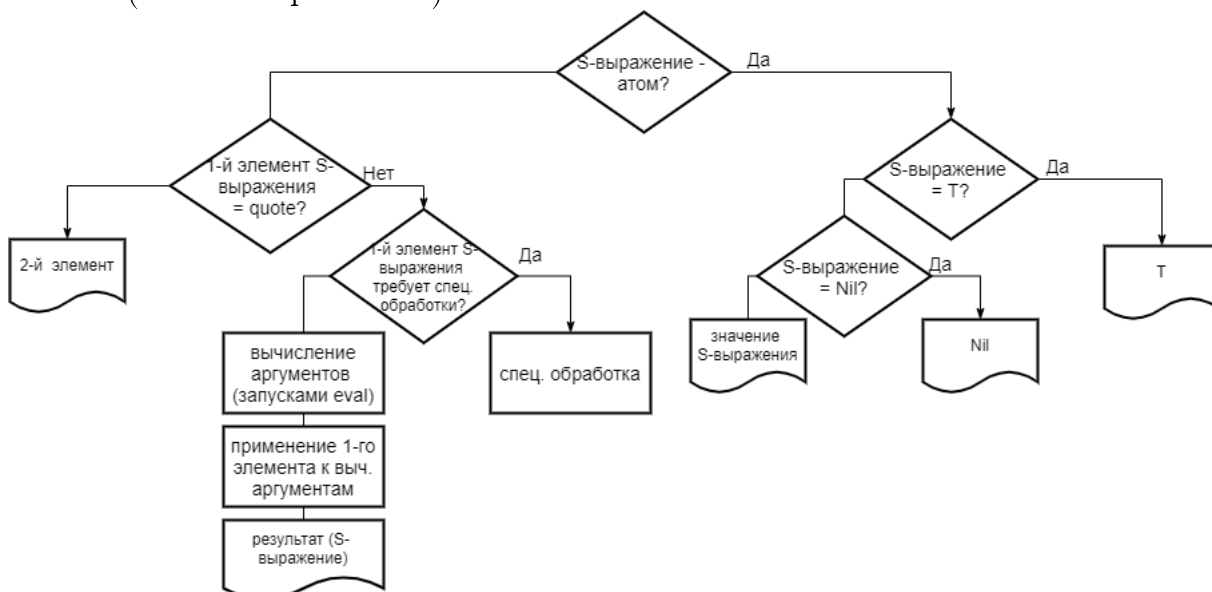
Итак, отличия: cons является базисной, list – нет; cons имеет фиксированное количество аргументов (два), list – произвольное; cons создает точечную пару или список (в зависимости от второго аргумента), list – список; в отличие от функции cons, результат функции list симметричен относительно аргументов:  $(list\ '(A)\ '(B)) \Rightarrow ((A)(B))$ , но  $(cons\ '(A)\ '(B)) \Rightarrow ((A)\ B)$ .

Cons эффективнее list, list определяется с помощью cons.

## Из указаний к выполнению работы

Eval выполняет двойное вычисление своего аргумента. Эта функция является обычной, и первое вычисление аргумента выполняет так же, как и любая обычная функция. Полученное при этом выражение вычисляется ещё раз. Такое двойное вычисление может понадобиться либо для снятия блокировки вычислений (установленной функцией quote), либо же для вычисления сформированного в ходе первого вычисления нового функционального вызова.

Вызов (eval S-выражение)



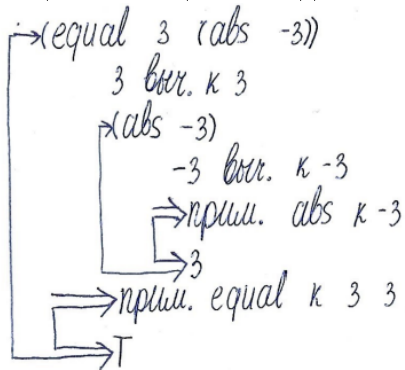
Квотирование объекта – это применение к нему функции `quote`, которая в качестве своего значения выдаёт сам аргумент, не вычисляя его. По сути, эта функция блокирует вычисление своего аргумента. Необходимость в этом нередко возникает при использовании обычных встроенных функций, чтобы задать аргументы в явном виде и избежать их вычисления. Константы–числа и атомы `T`, `NIL` при их использовании в качестве аргументов обычных функций можно не квотировать, поскольку значением любой константы является она сама. Функция `quote` используется часто, поэтому допускается упрощённый способ обращения к ней с помощью апострофа, маркирующего квотируемое выражение.

В диалекте `Common Lisp` для замыкания функционального аргумента встроена специальная форма `(function F)`, где `F` – определяющее выражение функции. Эту форму часто называют функциональной блокировкой, поскольку она аналогична по действию функции `quote`, но не просто квотирует аргумент, а как бы замыкает значения используемых в функциональном аргументе `F` свободных переменных, фиксируя их значения из контекста его определения. Функциональную блокировку можно записывать короче, с помощью двух знаков `#'` ("Получить функцию с данным именем").

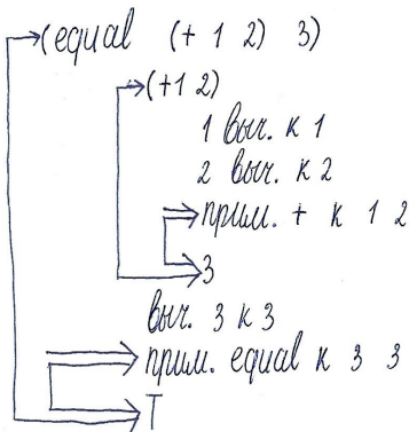
# Практические задания

1. Составить диаграмму вычисления следующих выражений:

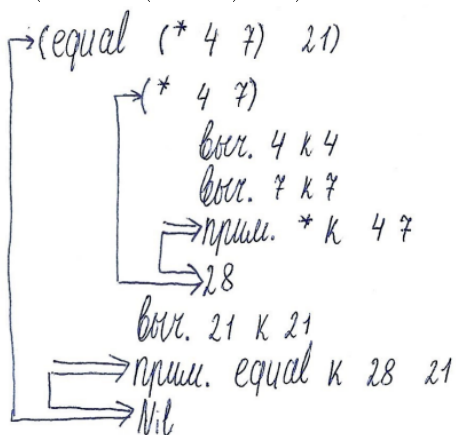
1. (equal 3 (abs - 3))



2. (equal (+ 1 2) 3)

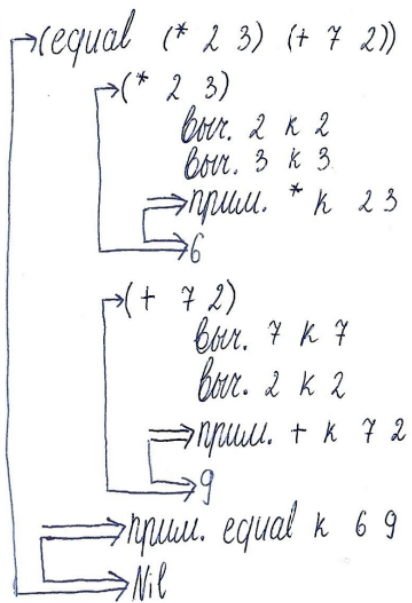


3. (equal (\* 4 7) 21)

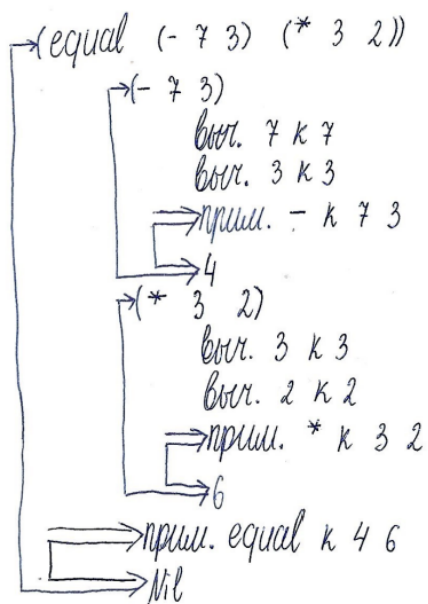




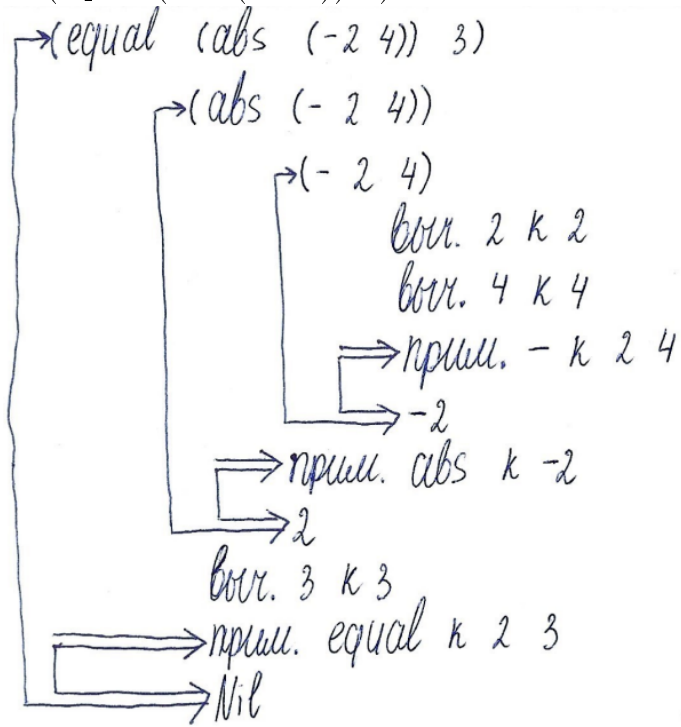
4. (equal (\* 2 3) (+ 7 2))



5. (equal (- 7 3) (\* 3 2))

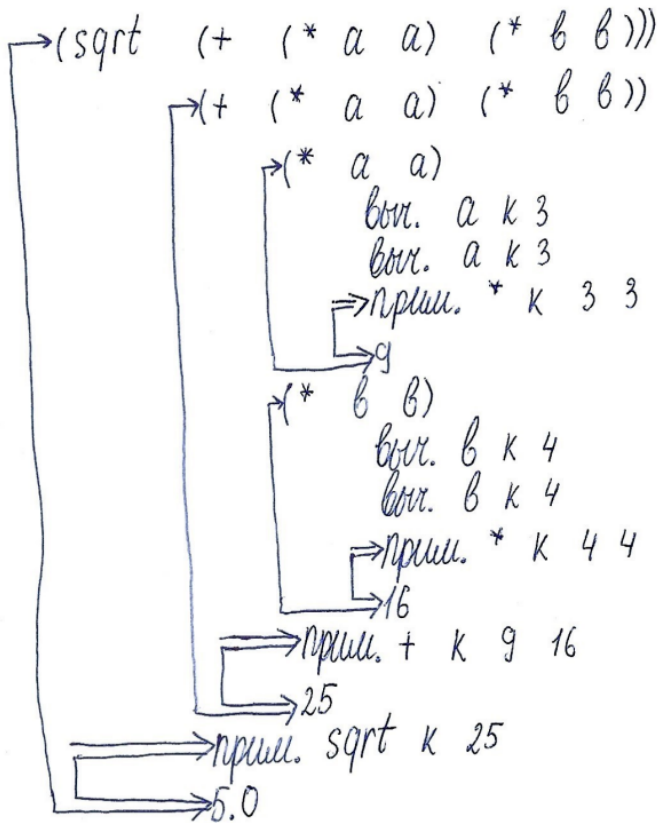


6. (equal (abs (- 2 4)) 3)



2. Написать функцию, вычисляющую гипотенузу прямоугольного треугольника по заданным катетам, и составить диаграмму её вычисления.

```
1 (defun hypotenuse (a b) (sqrt (+ (* a a) (* b b))))
2 (hypotenuse 3 4) => 5.0
```

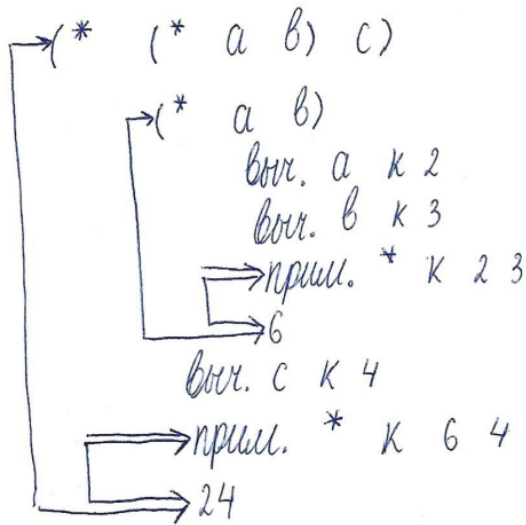


### 3. Написать функцию, вычисляющую объем параллелепипеда по 3-м его сторонам, и составить диаграмму ее вычисления.

```

1  (defun p_volume (a b c) (* (* a b) c))
2  (p_volume 2 3 4) => 24
3  ;; or
4  (defun p_volume (a b c) (* a b c))
5  (p_volume 2 3 4) => 24

```



### 4. Каковы результаты вычисления следующих выражений? (объяснить возможную ошибку и варианты ее устранения)

1.

```

1  (list 'a c) => The variable C is unbound.

```

Одна из возможных ошибок: переменная `c` не связана со значением. Решение: задать переменной `c` некоторое значение.

```

1  (let ((c 'c)) (list 'a c)) => (A C)

```

Другая из возможных ошибок: предполагалось, что `c` – это символ. Решение: использовать функцию `quote` (или сокращенную – апостроф) для блокировки вычисления аргумента.

```

1  (list 'a 'c) => (A C)

```

## 2.

```
1 (cons 'a (b c)) => Undefined function: B, Undefined variable: C
```

Одна из возможных ошибок: функция `b` не связана со своим определением, а переменная `c` не связана со своим значением. Решение: определить функцию `b` с одним аргументом (или переменным количеством аргументов), а переменной `c` задать некоторое значение.

```
1 (defun b (c) (cons 'b (cons c Nil)))  
2 (b 'c) => (B C)  
3 (let ((c 'c)) (cons 'a (b c))) => (A B C)
```

Другая из возможных ошибок: предполагалось, что `(b c)` – это список из символов `b` и `c`. Решение: использовать функцию `quote` (или сокращенную – апостроф) для блокировки вычисления аргументов.

```
1 (cons 'a '(b c)) => (A B C)
```

## 3.

```
1 (cons 'a '(b c)) => (A B C)
```

Ошибок нет

## 4.

```
1 (caddy (1 2 3 4 5)) => Undefined function: CADDY
```

Одна из возможных ошибок: функция `caddy` не связана со своим определением. Решение: определить функцию `caddy`, принимающую один аргумент (или переменное число аргументов), и использовать функцию `quote` (или сокращенную – апостроф) для блокировки вычисления аргумента.

```
1 (defun caddy (arg) arg)  
2 (caddy '(1 2 3 4 5)) => (1 2 3 4 5)
```

Другая из возможных ошибок: предполагалось вызвать функцию `caddr` для получения третьего элемента списка `(1 2 3 4 5)`. Решение: исправить опечатку (заменить `caddy` на `caddr`) и использовать функцию `quote` (или сокращенную – апостроф) для блокировки вычисления аргумента.

```
1 (caddr '(1 2 3 4 5)) => 3
```

## 5.

```
1 (cons 'a 'b 'c) => The function CONS is called with three arguments ,  
   but wants exactly two.
```

Ошибка: функция `cons` вызвана с тремя аргументами, хотя она принимает два аргумента. Решение: предполагая, что автор хотел получить список из трех символов (`a b c`), можно либо первым аргументом передать символ `a`, а вторым – список (`b c`), либо использовать функцию `list` вместо `cons`.

```
1 (cons 'a '(b c)) => (A B C)  
2 (list 'a 'b 'c) => (A B C)
```

## 6.

```
1 (list 'a (b c)) => Undefined function: B, Undefined variable: C
```

(Аналогично пункту 2)

Одна из возможных ошибок: функция `b` не связана со своим определением, а переменная `c` не связана со своим значением. Решение: определить функцию `b` с одним аргументом (или переменным количеством аргументов), а переменной `c` задать некоторое значение.

```
1 (defun b (c) (cons 'b (cons c Nil)))  
2 (b 'c) => (B C)  
3 (let ((c 'c)) (list 'a (b c))) => (A (B C))
```

Другая из возможных ошибок: предполагалось, что (`b c`) – это список из символов `b` и `c`. Решение: использовать функцию `quote` (или сокращенную – апостроф) для блокировки вычисления аргументов.

```
1 (list 'a '(b c)) => (A (B C))
```

## 7.

```
1 (list a '(b c)) => The variable A is unbound.
```

Одна из возможных ошибок: переменная `a` не связана со значением. Решение: задать переменной `a` некоторое значение.

```
1 (let ((a 'a)) (list a '(b c))) => (A (B C))
```

Другая из возможных ошибок: предполагалось, что `a` – это символ. Решение: использовать функцию `quote` (или сокращенную – апостроф) для блокировки вычисления аргумента.

```
1 (list 'a '(b c)) => (A (B C))
```

8.

```
1 (list (+ 1 '(length '(1 2 3)))) => Value of '(LENGTH '(1 2 3)) in  
  (+ 1 '(LENGTH '(1 2 3))) is (LENGTH '(1 2 3)), not a NUMBER.
```

Функция `+` в качестве аргументов ожидает аргументы типа `NUMBER`. Вторым аргументом ей передано `'(length '(1 2 3))`. Функция `length` возвращает длину переданного ей списка (тип `NUMBER`), однако апостроф блокирует вычисление. Таким образом, вместо длины списка типа `NUMBER` функции `+` в качестве второго аргумента передается значение `(LENGTH '(1 2 3))`. Решение - убрать апостроф, блокирующий вычисления аргумента.

```
1 (list (+ 1 (length '(1 2 3)))) => (4)
```

5. Написать функцию `longer_then` от двух списков-аргументов, которая возвращает `T`, если первый аргумент имеет большую длину

```
1 (defun longer_then (list1 list2) (> (length list1) (length list2)))  
2 (longer_then '(1 2 3) '(1 2)) => T  
3 (longer_then '(1 2) '(1 2 3)) => NIL  
4 (longer_then Nil '(1)) => NIL  
5 (longer_then '(1 2) '(1 2)) => NIL
```

6. Каковы результаты вычисления следующих выражений?

```
1 (cons 3 (list 5 6)) => (3 5 6)  
2 (cons 3 '(list 5 6)) => (3 LIST 5 6)  
3 (list 3 'from 9 'lives (- 9 3)) => (3 FROM 9 LIVES 6)  
4 (+ (length for 2 too)) (car '(21 22 23))) => The variable FOR is  
  unbound.  
5 (cdr '(cons is short for ans)) => (IS SHORT FOR ANS)  
6 (car (list one two)) => Undefined variables: ONE TWO  
7 (car (list 'one 'two)) => ONE
```

**7. Дана функция (defun mystery (x) (list (second x) (first x))). Какие результаты вычисления следующих выражений?**

Функции от SECOND до TENTH извлекают соответствующие элементы списка. LAST возвращает последнюю cons-ячейку в списке (если вызывается с целочисленным аргументом n, возвращает n ячеек).

```
1 (mystery (one two)) => The variable TWO is unbound
2 (mystery (last one two)) => The variable ONE is unbound
3 (mystery free) => The variable FREE is unbound
4 (mystery one 'two) => The variable ONE is unbound
```

Примеры корректной работы:

```
1 (mystery '(one two)) => (TWO ONE)
2 (mystery '(last one two)) => (ONE LAST)
3 (mystery (last '(one two))) => (NIL TWO)
4 (mystery '(free)) => (NIL FREE)
```

**8. Написать функцию, которая переводит температуру в системе Фаренгейта температуру по Цельсию (defun f-to-c (temp)...)**

Формулы:  $c = 5/9 * (f - 32.0)$ ;  $f = 9/5 * c + 32.0$ .

Как бы назывался роман Р. Брэдбери "451 по Фаренгейту" в системе по Цельсию?

```
1 (defun f-to-c (temp) (* (/ 5 9) (- temp 32.0)))
2 (f-to-c 451) => 232.77779
3
4 (defun c-to-f (temp) (+ (* (/ 9 5) temp) 32.0))
5 (c-to-f 232.77779) => 451.0
```

Ответ: "232.77779 по Цельсию"



## 9. Что получится при вычисления каждого из выражений?

```
1 (list 'cons t NIL) => (CONS T NIL)
2 (eval (list 'cons t NIL)) => (T)
3 (eval (eval (list 'cons t NIL))) => The function COMMON-LISP:T is
   undefined
4 ;(eval t) => T
5 (apply #cons "(t NIL)) => illegal complex number format: #CONS
6 ;(apply #'cons '(t NIL)) => (T)
7 (eval NIL) => NIL
8 (list 'eval NIL) => (EVAL NIL)
9 (eval (list 'eval NIL)) => NIL
10 ;(eval (list 'eval NIL)) = (eval (eval NIL)) = (eval NIL) => NIL
```

### Дополнительно

1. Написать функцию, вычисляющую катет по заданной гипотенузе и другому катету прямоугольного треугольника, и составить диаграмму ее вычисления.

```
1 (defun cathet (hyp another_cathet) (sqrt (- (* hyp hyp) (*
   another_cathet another_cathet))))
2 (cathet 5 4) => 3.0
```

