

CAR

EQUAL

CONS

CDR

ATOM

```
split-by (lst n)
  ((take (lst n)
    (if (or (= n 0) (null lst))
        '()
        (cons (car lst)
              (take (cdr lst) (- n 1))))))
  (cond
    ((<= n 0) lst)
    ((null lst) '())
    (t (cons (take lst n)
              (split-by (nthcdr n lst) n))))))
```

## Функциональное программирование: базовый курс

### Лекция 5. Анонимные функции и замыкания.

CAR EQUAL  
CONS  
CDR ATOM

Функциональное программирование: базовый курс

Лекция 5

Анонимные функции и замыкания

---

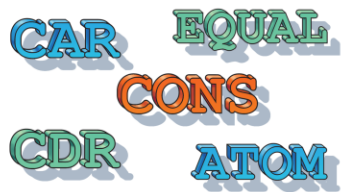
## Функции как объекты первого класса

CAR EQUAL  
CONS  
CDR ATOM

## Функции как объекты первого класса

---

- Функции являются объектами первого класса в языке, если их можно:
  - создавать на этапе выполнения программы
  - сохранять в переменных
  - передавать как аргументы другим функциям
  - возвращать из функции в качестве результата



## Косвенный вызов функции

---

```
[1]> (+ 1 20 21)
```

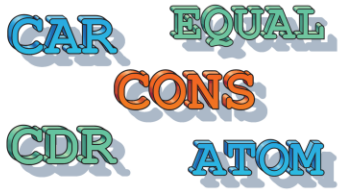
```
42
```

```
[2]> (funcall #' + 1 20 21)
```

```
42
```

```
[3]> (apply #' + 1 '(20 21))
```

```
42
```



```
[1]> (defparameter *fn* #'+)
```

```
*FN*
```

```
[2]> (*fn* 1 20 21)
```

; ошибка

```
*** - EVAL: undefined function *FN*
```

```
[3]> (funcall *fn* 1 20 21)
```

; ОК

```
42
```

```
[4]> (let ((fn #'+)
```

```
        (lst '(1 20 21))
```

```
        (apply fn lst))
```

```
42
```

CAR EQUAL  
CONS  
CDR ATOM

## Пример: построение графика функции

```
(defun f (x) (- (* x x) 1))
```

```
(defun func-coords (from to step)
  (do ((x from (+ x step)) res)
      ((> x to) (nreverse res))
      (push (list x (f x)) res)))
```

```
[1]> (func-coords -2.0 2.0 1.0)
((-2.0 3.0) (-1.0 0.0) (0.0 -1.0) (1.0 0.0) (2.0 3.0))
```

CAR EQUAL  
CONS  
CDR ATOM

(do (список-переменных)  
    (условие-остановки результат)  
    формы-тела-цикла)

```
[1]> (do ((i 0 (+ i 2)))  
        ((> i 6) (* i 100))  
        (format t "~a " i))
```

0 2 4 6

800

← результат, который  
возвращается формой do

CAR EQUAL  
CONS  
CDR ATOM

## Пример: построение графика функции

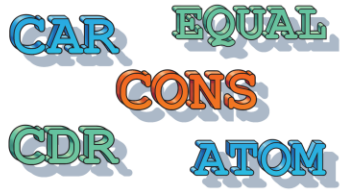
```
(defun f (x) (- (* x x) 1))
```

```
(defun g (x) (+ (* x x) 1))
```

```
(defun func-coords (fn from to step)
  (do ((x from (+ x step)) res)
      ((> x to) (nreverse res))
      (push (list x (funcall fn x)) res)))
```

```
[1]> (func-coords #'f -2.0 2.0 1.0)
((-2.0 3.0) (-1.0 0.0) (0.0 -1.0) (1.0 0.0) (2.0 3.0))
```





```
[1]> (defparameter *foo* 42)
```

```
*FOO*
```

```
[2]> (boundp '*foo*)
```

```
T
```

```
[3]> (fboundp '*foo*)
```

```
NIL
```

```
[4]> (print *foo*)
```

```
42
```

CAR EQUAL  
CONS  
CDR ATOM

```
[1]> (symbol-value '*foo*)
```

42

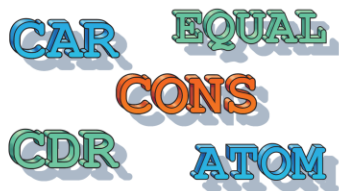
```
[2]> (setf (symbol-value '*foo*) 100)
```

100

```
[3]> (print *foo*)
```

100

здесь \*foo\* используется в  
качестве переменной



```
[1]> (defun foo-func (x y) (+ x y))  
FOO-FUNC
```

```
[2]> (setf (symbol-function '*foo*)  
          #'foo-func)  
#<FUNCTION FOO-FUNC (X Y)>
```

здесь `*foo*` используется в качестве функции

```
[3]> (*foo* *foo* *foo*)  
200
```

A diagram with two red boxes. The first box contains the text `*foo*` and has a vertical line pointing up to the text 'здесь \*foo\* используется в качестве функции' (here \*foo\* is used as a function). The second box contains the text `*foo* *foo*` and has a vertical line pointing down to the text 'здесь \*foo\* используется в качестве переменной' (here \*foo\* is used as a variable).

здесь `*foo*` используется в качестве переменной

CAR EQUAL  
CONS  
CDR ATOM

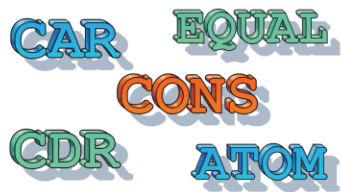
```
[1]> (funcall (function +) 20 22)
```

42

```
[2]> (funcall #' + 20 22)
```

42

раскрывается в (function +)



Функциональное программирование: базовый курс

Лекция 5

Анонимные функции и замыкания

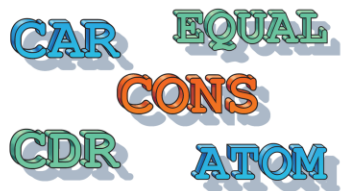
---

## Анонимные функции

CAR EQUAL  
CONS  
CDR ATOM

```
[1]> (lambda (x) (- (* x x) 1))  
#<FUNCTION :LAMBDA (X) (- (* X X) 1)>
```

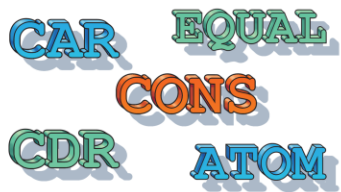
```
[2]> (func-coords  
      #'(lambda (x) (- (* x x) 1))  
      -2.0 2.0 1.0)  
( (-2.0 3.0) (-1.0 0.0) (0.0 -1.0) (1.0 0.0) (2.0 3.0) )
```



```
[1]> (setf f  
      #'(lambda (x) (- (* x x) 1)))  
#<FUNCTION :LAMBDA (X) (- (* X X) 1)>
```

```
[2]> (funcall f 2.0)  
3.0
```

```
[3]> (funcall  
      #'(lambda (x) (- (* x x) 1))  
      2.0)  
3.0
```



## Реализация макроса defun на основе lambda

```
[1]> (macroexpand '(defun foo (x) (+ x 3)))
```

```
(LET NIL (SYSTEM::REMOVE-OLD-DEFINITIONS 'FOO) (SYSTEM::EVAL-WHEN-COMPILE (SYSTEM::C-DEFUN 'FOO (SYSTEM::LAMBDA-LIST-TO-SIGNATURE '(X))))
```

```
(SYSTEM::%PUTD 'FOO (FUNCTION FOO (LAMBDA (X) (DECLARE (SYSTEM::IN-DEFUN FOO)) (BLOCK FOO (+ X 3)))))
```

```
(EVAL-WHEN (EVAL) (SYSTEM::%PUT 'FOO 'SYSTEM::DEFINITION (CONS '(DEFUN FOO (X) (+ X 3)) (THE-ENVIRONMENT)))) 'FOO ;
```

T



CAR EQUAL  
CONS  
CDR ATOM

- λ-исчисление
  - формальная система, предложенная в 1936 году математиком Алонзо Чёрчем
  - является примитивным чистым функциональным языком программирования
  - элементы λ-исчисления были использованы в 1958 году Джоном Маккарти при реализации Лиспа



CAR EQUAL  
CONS  
CDR ATOM

$$g \circ f = g( f( x ) )$$


```
(defun compose (f g)
  (lambda (x)
    (funcall g (funcall f x)))))

[1]> (let ((as (compose #'abs #'sin)))
      (funcall as -1.0))

0.84147096
```

CAR EQUAL  
CONS  
CDR ATOM

передается в качестве данных



```
((lambda (s)
  (print (list s (list 'quote s)))))
'(lambda (s)
  (print (list s (list 'quote s)))))
```

CAR EQUAL  
CONS  
CDR ATOM

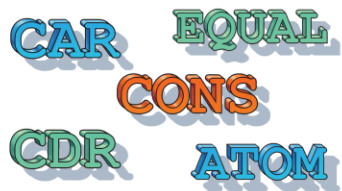
Функциональное программирование: базовый курс

Лекция 5

Анонимные функции и замыкания

---

## Замыкания



## Задача: получить случайный элемент последовательности

---

```
(defun random-elt (seq)
  (elt seq (random (length seq))))
```

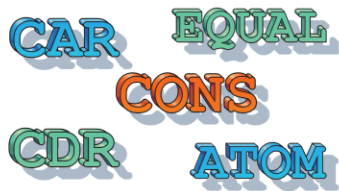
```
[1]> (defparameter *list1* '(#\a #\B #\c #\D))
*LIST1*
```

```
[2]> (defparameter *list2* '(10 20 30 40 50))
*LIST2*
```

```
[3]> (random-elt *list1*)
#\B
```

```
[4]> (random-elt *list1*)
#\c
```

```
[5]> (random-elt *list2*)
50
```



## Задача: получить случайный элемент последовательности

```
(defun make-random-gen (seq)
  (let ((seq-len (length seq)))
    (lambda ()
      (elt seq (random seq-len))))))
```

```
[1]> (defparameter *rand-char*
      (make-random-gen *list1*))
```

\*RAND-CHAR\*

```
[2]> (defparameter *rand-num*
      (make-random-gen *list2*))
```

\*RAND-NUM\*

```
[3]> (funcall *rand-char*)
```

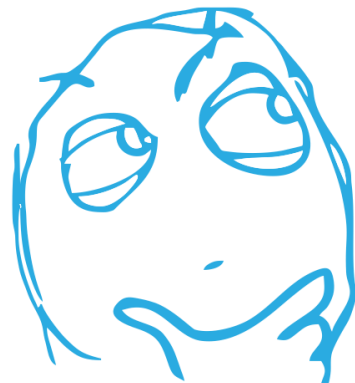
#\a

```
[4]> (funcall *rand-num*)
```

20

```
[5]> (funcall *rand-num*)
```

50



CAR EQUAL  
CONS  
CDR ATOM

funarg = function argument

```
(defun make-random-gen (seq)
  (let ((seq-len (length seq)))
    (lambda ()
      (elt seq (random seq-len))))))
```

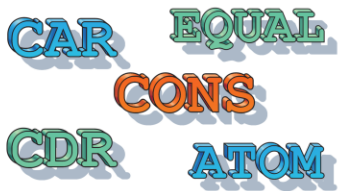
CAR EQUAL  
CONS  
CDR ATOM

- Замыкание (closure) – функция, которая ссылается на локальные переменные, определенные вне тела этой функции



Инкапсуляция – связывание данных с кодом, который их обрабатывает





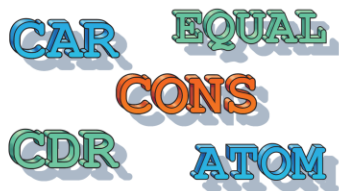
## Совместное использование данных замыканиями

- Совместное использование данных (data sharing) – доступ к одним и тем же переменным из разных замыканий

```
(defun make-closures (arg)
  (values
    (lambda ()
      (incf arg)
      (format t "thumbs up! ~c~%" (code-char #x1f44d)) arg)
    (lambda ()
      (decf arg)
      (format t "thumbs down! ~c~%" (code-char #x1f44e)) arg))))
```

```
[1]> (multiple-value-bind (up down) (make-closures 42)
      (funcall up) (funcall down) (funcall up))
```

thumbs up! 👍  
thumbs down! 👎  
thumbs up! 👍



- Частичное применение функции (partial application) – фиксация одного из аргументов функции и создание функции с меньшим количеством аргументов

```
(defun f (x y z) (+ x y z))  
(defun g (x y z) (* x y z))  
(defun h (x y z) (list x y z))
```

```
(defun twisted-func (f1 f2 f3 a b)  
  (funcall f3  
    (funcall f1 a b) (funcall f2 a b)))
```

CAR EQUAL  
CONS  
CDR ATOM

```
(defun papply (fn fst-arg)
  (lambda (&rest other-args)
    (apply fn (cons fst-arg other-args)))))
```

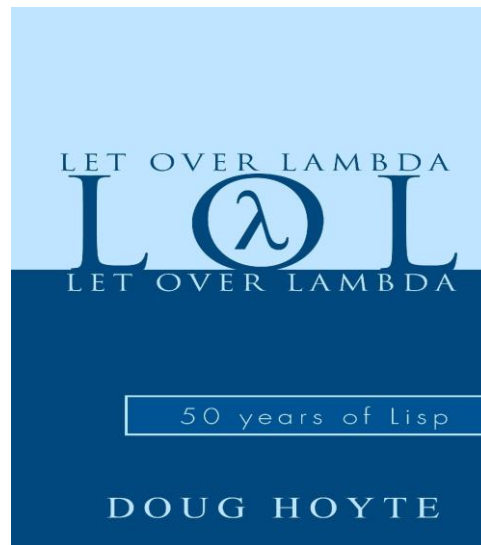
```
[1]> (twisted-func
      (papply #'f 1) ; (f 1 ? ?)
      (papply #'g 2) ; (g 2 ? ?)
      (papply #'h 3) ; (h 3 ? ?)
      10 20)
(3 31 400)
```

CAR EQUAL  
CONS  
CDR ATOM

## let over lambda

```
(let (переменные)
  (lambda (аргументы)
    ... ))
```

```
(let (переменные)
  (defun имя (аргументы)
    ... ))
```



"Let Over Lambda – 50 years of Lisp" –  
знаменитая книга Дуга Хойта  
о программировании макросов на Лиспе

## Задача: генератор чисел Фибоначчи

CAR EQUAL  
CONS  
CDR ATOM

```
(defun fib-next ()  
  ...)
```

```
[1]> (fib-next)      ; первое число ряда  
1
```

```
[2]> (fib-next)      ; второе число ряда  
1
```

```
[3]> (fib-next)      ; и т.д.  
2
```

```
[4]> (fib-next)  
3
```

```
...
```

## Задача: генератор чисел Фибоначчи

CAR EQUAL  
CONS  
CDR ATOM

```
(let ((p 1) (n 1))  
  (defun fib-next ()  
    (let (res)  
      (shiftf res p n (+ p n))  
      res)))
```

```
[1]> (fib-next)
```

```
1
```

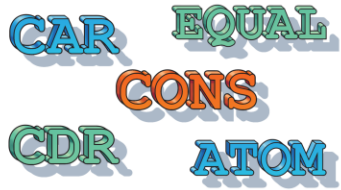
```
[2]> (fib-next)
```

```
1
```

```
[3]> (fib-next)
```

```
2
```

```
...
```



- что такое объекты первого класса
- что такое функции высшего порядка
- чем отличается вызов функции с помощью funcall от вызова с помощью apply
- что такое лямбда-выражения и как на их основе создаются анонимные функции
- что такое замыкания, как они создаются и для чего применяются