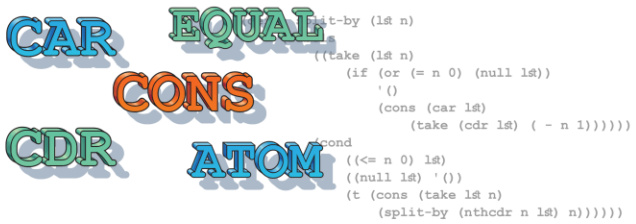
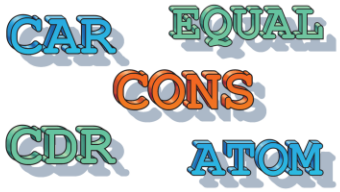


Функциональное программирование: базовый курс

Лекция 3.

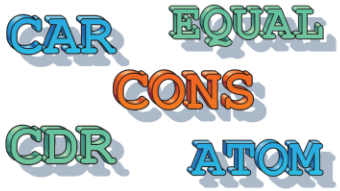
Работа с массивами и списками в языке Lisp.



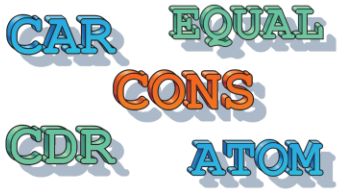
Лекция 3

Работа с массивами и списками в языке Lisp

Массивы



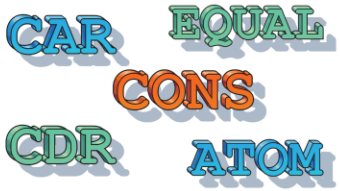
- Массив – упорядоченное множество элементов, обращение к которым выполняется по их порядковым номерам (индексам)
  - тип элементов
  - хранение элементов в памяти
  - время доступа к элементам
  - размерность
  - диапазоны индексов



Создание массива в Лиспе

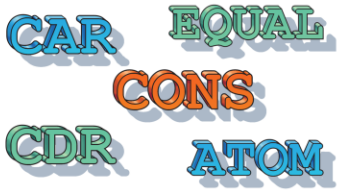
(make-array размерность параметры)

```
[1]> (make-array 5)
#(0 0 0 0 0)

[2]> (make-array '(3 3))
#2A((0 0 0) (0 0 0) (0 0 0))
```



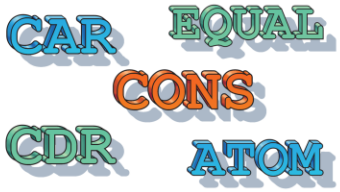
Доступ к элементам массива

```
[1]> (defvar a (make-array '(3 3)))
A
[2]> (aref a 1 1)       ; в Си a[1][1]
0
[3]> (setf (aref a 1 1) 42)
42
[4]> a
#2A((0 0 0) (0 42 0) (0 0 0))
```



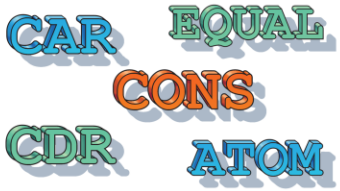
Инициализация элементов массива

```
[1]> (setf a (make-array 3
             :initial-element 42))
#(42 42 42)

[2]> (setf a (make-array 5
             :initial-contents '(1 2 3 4 5)))
#(1 2 3 4 5)

[3]> (setf a (vector 11 22 33 44 55))
#(11 22 33 44 55)
```



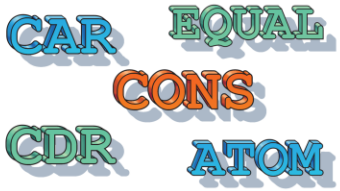
Информация о массиве

```
[1]> (defvar a #2A(
                  (1 #\2)
                  ("3" '(4 4 4 4))))
A
[2]> (arrayp a)
T
[3]> (array-rank a)
2
[4]> (array-dimensions a)
(2 2)
```



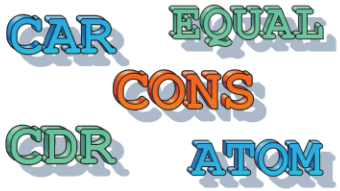
Статические и динамические массивы

```
[1]> (defvar arr (make-array 3
                  :initial-contents '(1 2 3)))
ARR

[2]> (setf arr (adjust-array arr 6))
#(1 2 3 0 0 0)
```



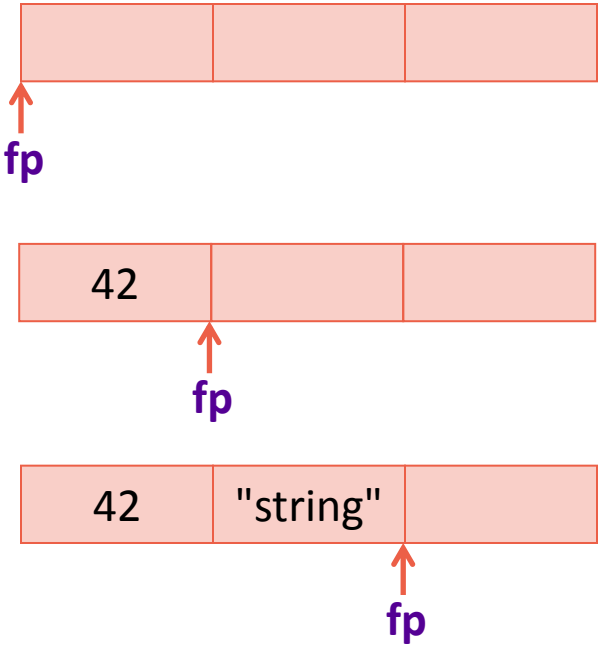
Размер массива

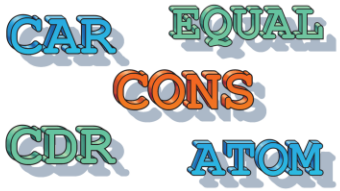
```
[1]> (setf arr (make-array '(3 3 3)))
#3A(((0 0 0) (0 0 0) (0 0 0))
    ((0 0 0) (0 0 0) (0 0 0))
    ((0 0 0) (0 0 0) (0 0 0)))

[2]> (array-total-size arr)
27
```



Одномерные массивы с указателем заполнения

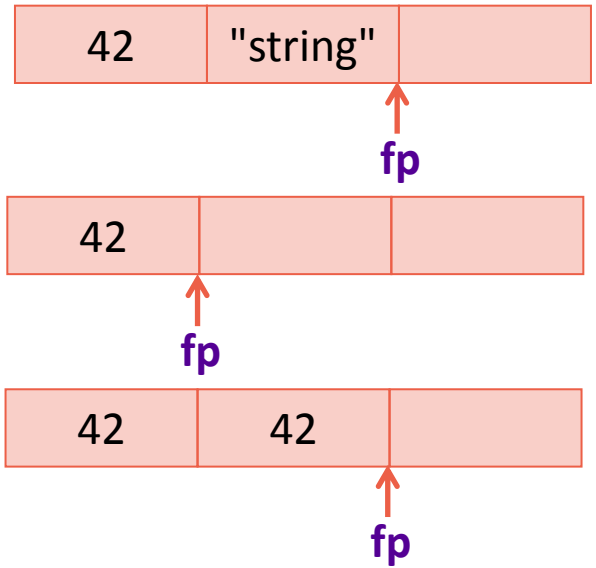
```
[1]> (setf arr
           (make-array 3 :fill-pointer 0))
#()

[2]> (vector-push 42 arr)
0

[3]> (vector-push "string" arr)
1
```

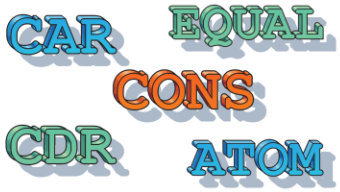




Одномерные массивы с указателем заполнения

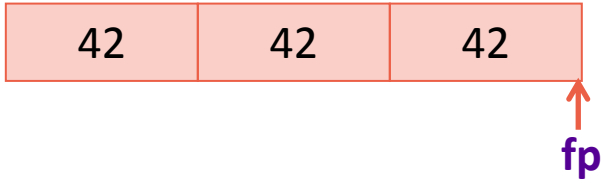
```
[4]> arr
#(42 "string")

[5]> (vector-pop arr)
"string"

[6]> (vector-push 42 arr)
1
[7]> (fill-pointer arr)
2
```



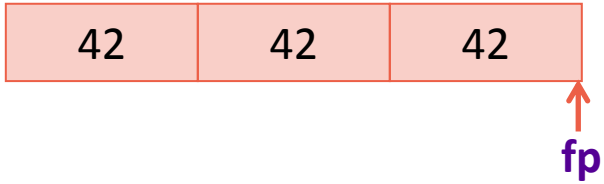


Одномерные массивы с указателем заполнения

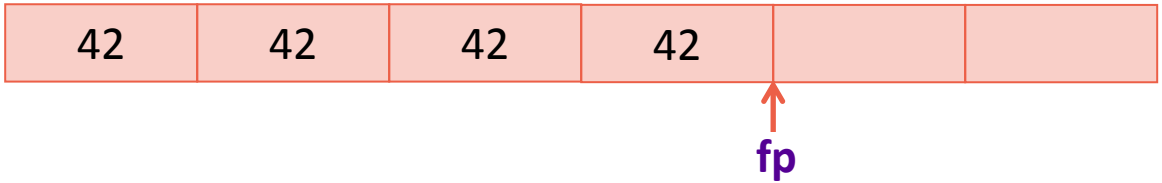
```
[8]> (vector-push 42 arr)
2
```

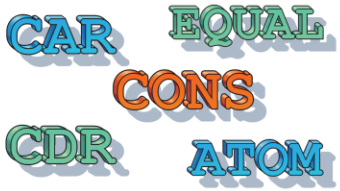



```
[9]> (vector-push 42 arr)
NIL
```




```
[10]> (vector-push-extend
                    42 arr 3)
3
[11]> arr
#(42 42 42 42)
```

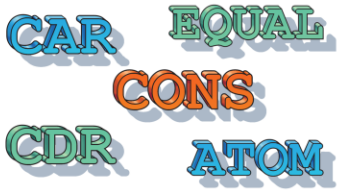




Задача: поиск элемента в многомерном массиве

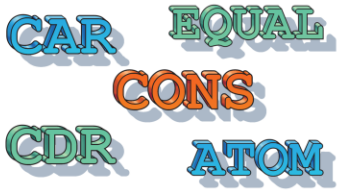
Найти заданный элемент в многомерном массиве (определить индексы)

```
[1]> (defvar hyper (make-array '(5 5 5 5)))
HYPER

[2]> (setf (aref hyper
                    (random 5) (random 5)
                    (random 5) (random 5))
                42)
42
```



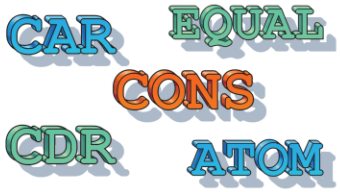
Задача: поиск элемента в многомерном массиве

```
(dotimes (i 5)
  (dotimes (j 5)
    (dotimes (k 5)
      (dotimes (s 5)
        (when (= 42 (aref hyper i j k s))
          вернуть_список_индексов)))))
```



Задача: поиск элемента в многомерном массиве

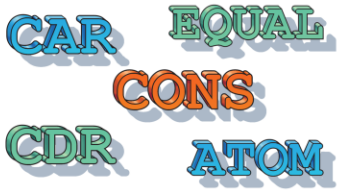
```
(block outer
(dotimes (i 5)
  (dotimes (j 5)
    (dotimes (k 5)
      (dotimes (s 5)
        (when (= 42 (aref hyper i j k s))
          (return-from outer
            (list i j k s))))))))
```



Лекция 3

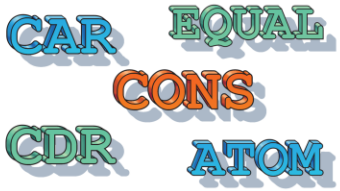
Работа с массивами и списками в языке Lisp

Работа со строками



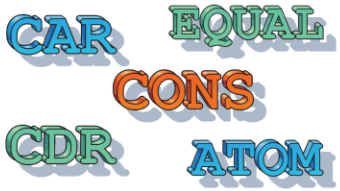
Создание строки

```
[1]> (defvar str "flip")
STR
[2]> (arrayp str)
T
[3]> (aref str 3)
#\p
[4]> (char str 3)
#\p
[5]> (setf (aref str 2) #\o)
#\o
[6]> str
"flop"
```



Длина строки

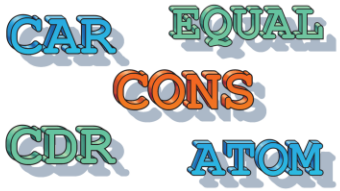
```
[7]> str
"flop"
[8]> (length str)
4
[9]> (array-total-size str)
4
[10]> (array-rank str)
1
[11]> (array-dimensions str)
(4)
```



Копирование строк

```
[1]> (defvar s1 "flip")
S1
[2]> (defvar s2)
S2
[3]> (setf s2 s1)
"flip"
[4]> (setf (aref s2 2) #\o)
#\o
[5]> s1
"flop"
[6]> s2
"flop"
[7]> (eq s1 s2)
T
```

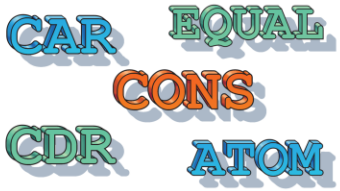
```
[1]> (defvar s1 "flip")
S1
[2]> (defvar s2)
S2
[3]> (setf s2 (copy-seq s1))
"flip"
[4]> (setf (aref s2 2) #\o)
#\o
[5]> s1
"flip"
[6]> s2
"flop"
[7]> (eq s1 s2)
NIL
```



Конкатенация строк

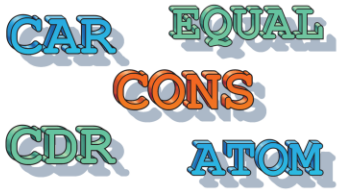
"flip" + "flop" = "flipflop"

```
[1]> (setf s1 "flip" s2 "flop")
"flop"
[2]> (concatenate 'string s1 s2)
"flipflop"
```



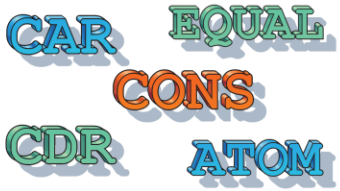
Изменение регистра символов и обращение строки

```
[1]> (string-upcase "high")
"HIGH"
[2]> (string-downcase "LOW")
"low"
[3]> (string-capitalize "cAMEL cASE")
"Camel Case"
[4]> (reverse "нофелет")
"телефон"
```



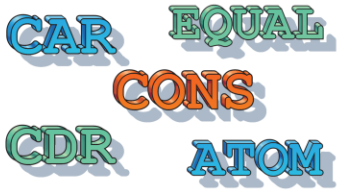
Сравнение строк

```
[1]> (string-equal "itmo" "ITMO")
T
[2]> (string= "itmo" "ITMO")
NIL
[3]> (string> "ITMO" "IFMO")
1
[4]> (string< "it" "itmo")
2
```



Поиск символов и подстрок

```
[1]> (position #\T "ITMO University")
1
[2]> (position #\t "ITMO University")
13
[3]> (position #\f "ITMO University")
NIL
[4]> (search "it" "ITMO University")
12
[5]> (search "IT" "ITMO University")
0
[6]> (search "if" "ITMO University")
NIL
```

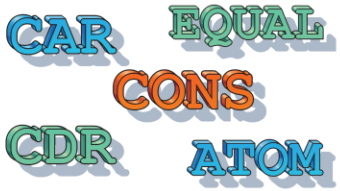


Удаление символов из строки

```
[1]> (remove #\. "I.T.M.O.")
"ITMO"
[2]> (string-trim "#$@%!" "#$#%!42$#@$#!!")
"42"
[3]> (string-left-trim " " "     42")
"42"
[4]> (string-right-trim
           "!? :-)"
           "Hi!! :-)))))" )
"Hi"
[5]> (substitute #\  #\. "I.T.M.O.")
"I T M O"
```

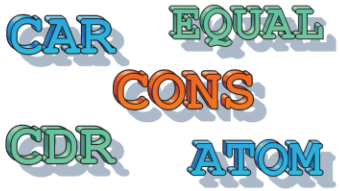
что заменять

на что заменять



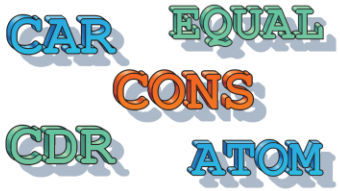
Преобразование чисел в строки

```
[1]> (write-to-string 42)
"42"

[2]> (format nil
           "~a is picked randomly"
           (random 1000))
"655 is picked randomly"
```



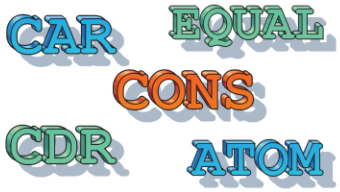
Преобразование строк в числа

```
[1]> (parse-integer "42")
42;
2
[2]> (parse-integer
                "42 is the Answer")
*** - PARSE-INTEGER: ERROR

[3]> (parse-integer
                "42 is the Answer"
                :junk-allowed t)
42;
2
```



Преобразование строк в числа

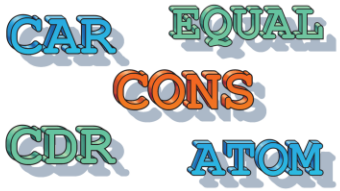
```
[1]> (read-from-string "3.14")
3.14;
4
[2]> (read-from-string "#x2A !")
42;
5
[3]> (read-from-string "1/42 ?")
1/42;
5
[4]> (setf x (read-from-string "1/42"))
X
[5]> (numberp x)
T
```



Лекция 3

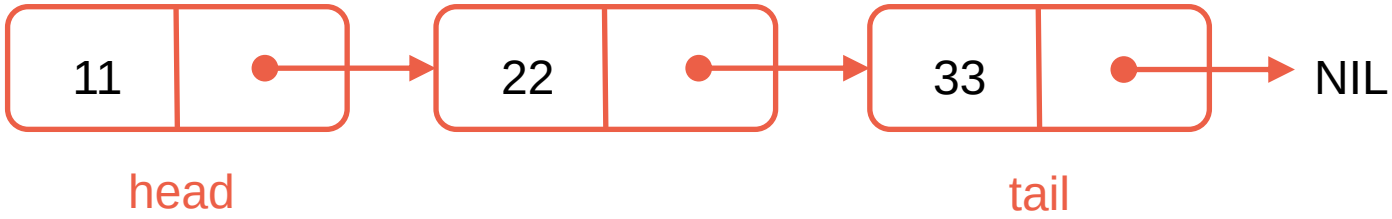
Работа с массивами и списками в языке Lisp

Списки

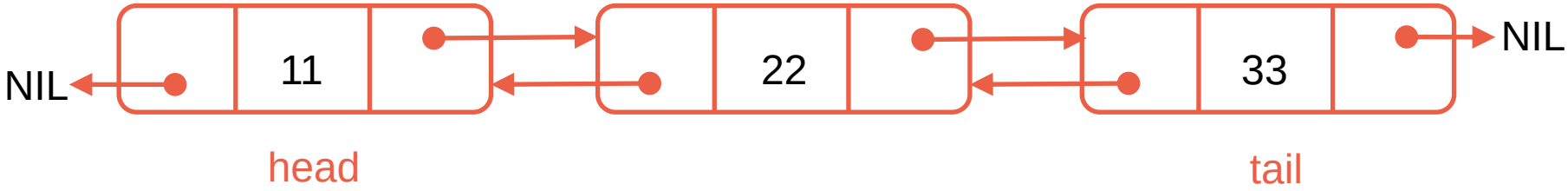


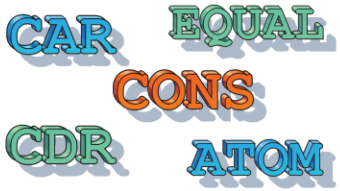
Список как абстрактный тип данных

- Односвязный список (singly linked list)



- Двусвязный список (doubly linked list)

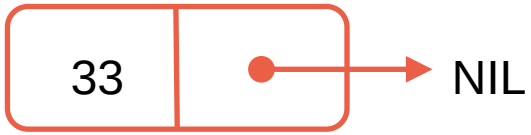




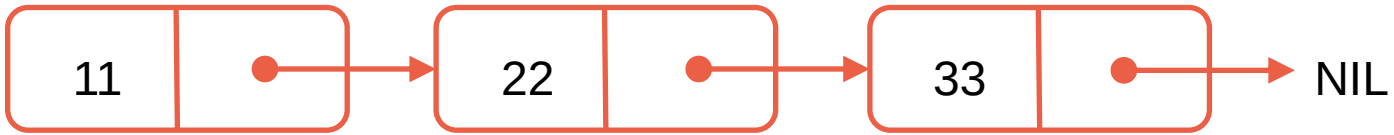
Список в Лиспе

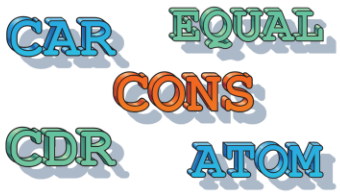
- Составлен из cons-ячеек (cons cells)

```
[1]> (cons 33 nil)
(33)
```



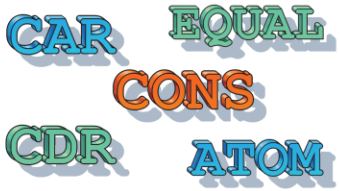

```
[2]> (cons 11 (cons 22 (cons 33 nil)))
(11 22 33)
```





Литерал списка

```
[1]> '(11 22 33)
(11 22 33)
[2]> (quote (11 22 33))
(11 22 33)
[3]> (list 11 22 33)
(11 22 33)
[4]> (list)
NIL
```

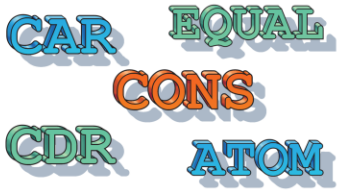


Пустой список

```
[1]> ()
NIL
[2]> '()
NIL
[3]> nil
NIL
[4]> 'nil
NIL
```

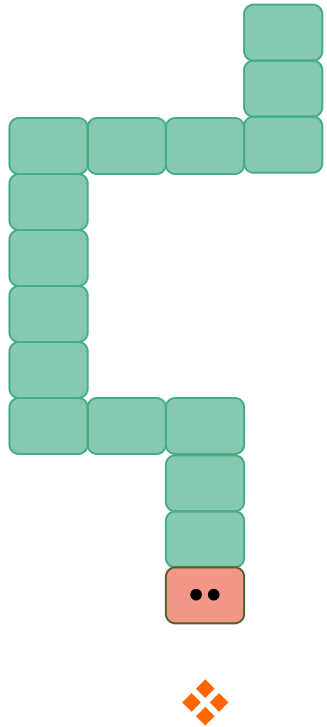
() = '() = nil = 'nil

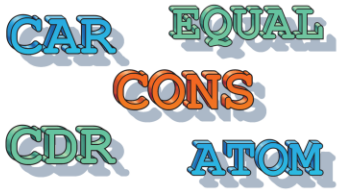
пустой список – единственное ложное значение в Лиспе



«Голова» и «хвост» списка

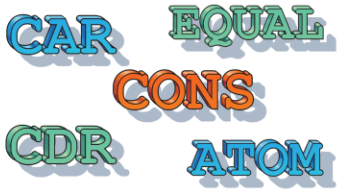
```
[1]> (defvar lst '(11 22 33))
LST
[2]> (car lst)
11
[3]> (cdr lst)
(22 33)
[4]> (first lst)
11
[5]> (rest lst)
(22 33)
```





Доступ к элементам списка

```
[1]> (defvar fruits
                '(banana apple orange))
FRUITS
[2]> (car fruits)
BANANA
[3]> (car (cdr fruits))
APPLE
[4]> (car (cdr (cdr fruits)))
ORANGE
[5]> (caddr fruits)
ORANGE
```



Доступ к элементам списка

- car, cdr, cadr, cddr, …, cddddr

```
[1]> (setf lst '(((1 2) ((3) (4 5)) 6)))
(((1 2) ((3) (4 5)) 6))
[2]> (caddar lst)
6
```

- first, second, third, …, tenth

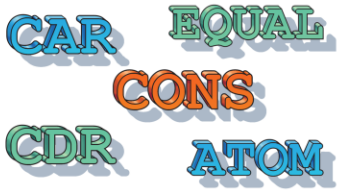
```
[3]> (third (car lst))
6
```

- nth

```
[4]> (nth 2 (car lst))
6
```

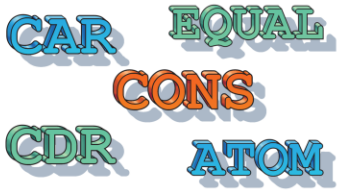
- elt

```
[5]> (elt (car lst) 2)
6
```



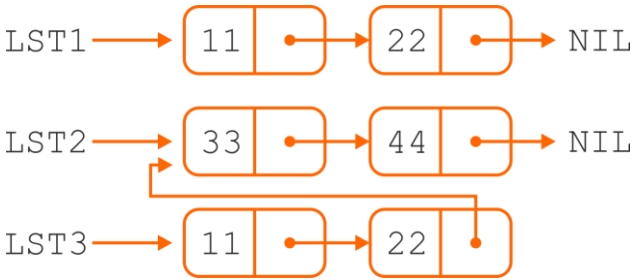
Замена элемента списка

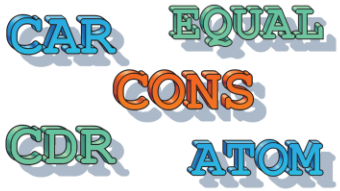
```
[1]> (defvar lst '(11 22 33))
LST
[2]> (setf (nth 1 lst) 44)
44
[3]> lst
(11 44 33)
```



Объединение списков: append

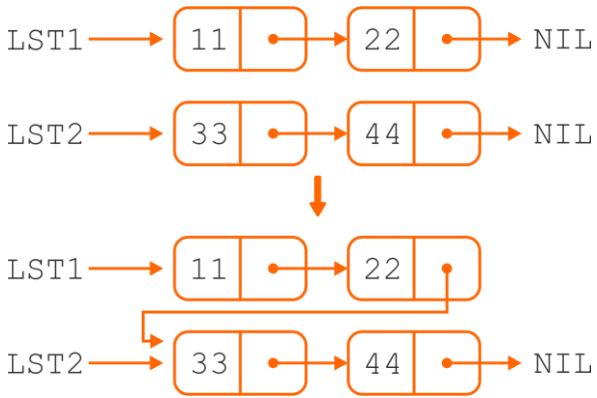
```
[1]> (defvar lst1 '(11 22))
LST1
[2]> (defvar lst2 '(33 44))
LST2
[3]> (defvar lst3
              (append lst1 lst2))
LST3


[4]> lst3
(11 22 33 44)
[5]> (eq (cddr lst3) lst2)
T
```



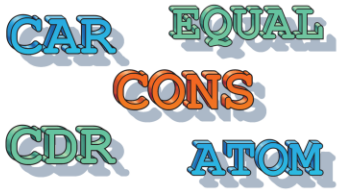


Объединение списков: nconc

```
[1]> (defvar lst1 '(11 22))
LST1
[2]> (defvar lst2 '(33 44))
LST2
[3]> (nconc lst1 lst2)
(11 22 33 44)
```



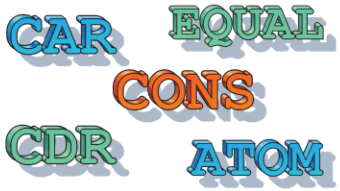

```
[4]> lst1
(11 22 33 44)
```



Разрушающие и неразрушающие функции

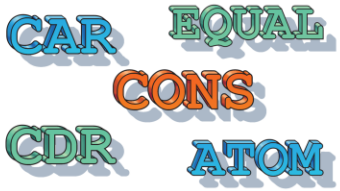
```
[1]> (defvar lst '(11 22 11 44))
LST
[2]> (remove 11 lst)
(22 44)
[3]> lst
(11 22 33 44)
[4]> (delete 11 lst)
(22 44)
[5]> lst
(11 22 44)
[6]> (setf lst
               (delete 11 lst))
(22 44)
[7]> lst
(22 44)
```

функция	разрушающий аналог
append	nconc
revappend	nreconc
reverse	nreverse
remove	delete
butlast	nbutlast
set-difference	nset-difference
set-exclusive-or	nset-exclusive-or
union	nunion
intersection	nintersection



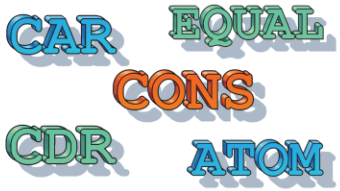
Добавление элементов в список: push/nreverse

```
[1]> (setf lst '(11))
(11)
[2]> (push 22 lst)
(22 11)
[3]> (push 33 lst)
(33 22 11)
[4]> lst
(33 22 11)
[5]> (setf lst (nreverse lst))
(11 22 33)
```



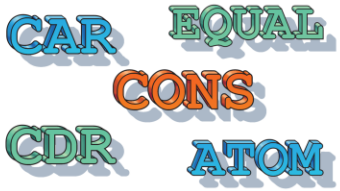
Добавление элементов в список: nconc

```
[1]> (setf lst '(11))
(11)
[2]> (nconc lst '(22))
(11 22)
[3]> (nconc lst (cons 33 nil))
(11 22 33)
[4]> lst
(11 22 33)
```



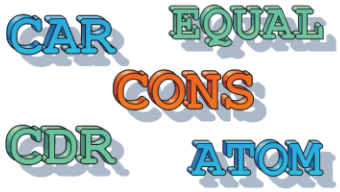
Добавление элемента к пустому списку

```
[1]> (setf lst1 nil lst2 nil)
NIL
[2]> (push 11 lst1)
(11)
[3]> lst1
(11)
[4]> (nconc lst2 (cons 11 nil))
(11)
[5]> lst2
NIL
[6]> (setf lst2 (nconc lst2 (cons 11 nil)))
(11)
[7]> lst2
(11)
```



Добавление элементов в список: pushnew

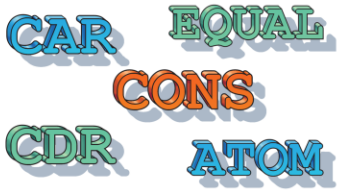
```
[1]> (setf lst '(11 22 33))
(11 22 33)
[2]> (pushnew 22 lst)
(11 22 33)
[3]> (pushnew 44 lst)
(44 11 22 33)
[4]> (pop lst)
44
[5]> lst
(11 22 33)
```



Лекция 3

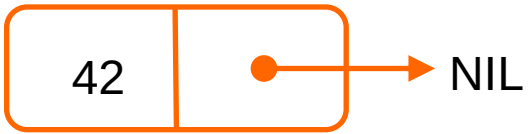
Работа с массивами и списками в языке Lisp

Ассоциативные списки и множества

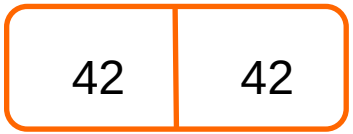


Точечные пары

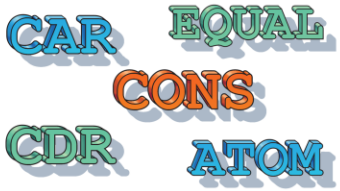
```
[1]> (cons 42 nil)
(42)
```




```
[2]> (cons 42 42)
(42 . 42)
```




```
[3]> '(42 . 42)
(42 . 42)
```

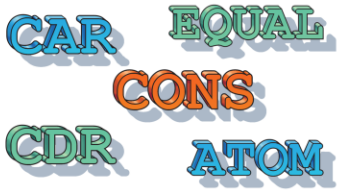


Ассоциативные списки

Ассоциативный список – простейшая реализация таблицы поиска (lookup table)

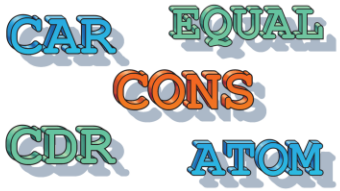
ключ	значение
1	"one"
2	"two"
...	...

```
[1]> (setf *alist*
           '((1 . "one") (2 . "two")))
((1 . "one") (2 . "two"))
```



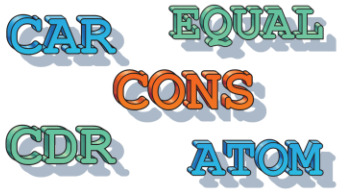
Поиск в ассоциативных списках

```
[1]> (setf *alist*
                '((1 . "one") (2 . "two")))
((1 . "one") (2 . "two"))

[2]> (assoc 2 *alist*)
(2 . "two")

[3]> (assoc 3 *alist*)
NIL
```



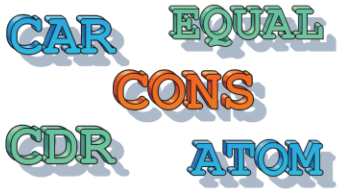
Поиск в ассоциативных списках

```
[1]> (setf *alist*
           '(("one" . 1) ("two" . 2)))
(("one" . 1) ("two" . 2))

[2]> (assoc "two" *alist*)
NIL

[3]> (assoc "two" *alist*
            :test #'string=)
("two" . 2)
```



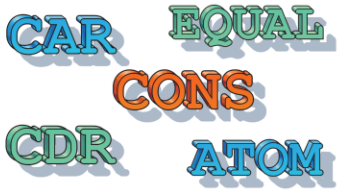
Добавление пар в ассоциативный список

```
[1]> (setf *alist*
           (cons (cons "three" 3) *alist*)
(("three" . 3)("one" . 1) ("two" . 2))

[2]> (push (cons "four" 4) *alist*)
(("four" . 4)("three" . 3)("one" . 1) ("two" . 2))

[3]> (setf *alist*
           (acons "five" 5 *alist*))
(("five" . 5)("four" . 4)("three" . 3)("one" . 1) ("two" . 2))
```



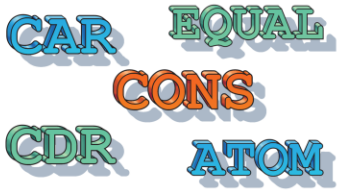
Пары с одинаковыми ключами в ассоциативном списке

```
[1]> ((setf *alist*
                '(("one" . 1) ("two" . 2)))
(("one" . 1) ("two" . 2))

[2]> (push '("two" . 22) *alist*)
(("two" . 22)("one" . 1) ("two" . 2))

[3]> (assoc "two" *alist* :test #'string=)
("two" . 22)
```



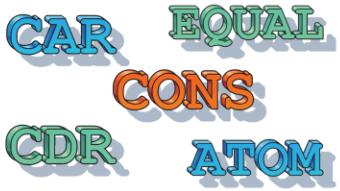
Создание ассоциативного списка из двух списков

```
[1]> (setf *list1*
           '("one" "two" "three"))
("one" "two" "three")

[2]> (setf *list2* '(1 2 3))
(1 2 3)

[3]> (setf *alist*
           (pairlis *list1* *list2*))
(("three" . 3) ("two" . 2) ("one" . 1))
```



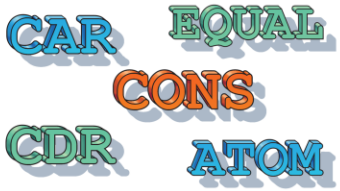
Обратный поиск в ассоциативном списке

```
[1]> ((setf *alist*
                '(("one" . 1) ("two" . 2)))
(("one" . 1) ("two" . 2))

[2]> (rassoc 2 *alist*)
("two" . 2)
```



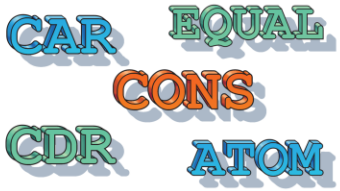
Список свойств

```
[1]> (setf *plist* '(one 1 two 2))
(ONE 1 TWO 2)

[2]> (getf *plist* 'two)
2

[3]> (setf (getf *plist* 'two) 22)
22

[4]> *plist*
(ONE 1 TWO 22)
```



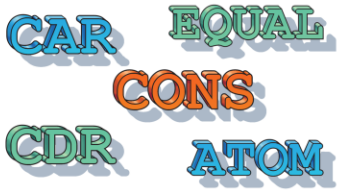
Использование списков как множеств

```
[1]> (defvar *set* nil)
*SET*
[2]> (setf *set* (adjoin 1 *set*))
(1)
[3]> (setf *set* (adjoin 2 *set*))
(2 1)
[4]> (setf *set* (adjoin 1 *set*))
(2 1)
```



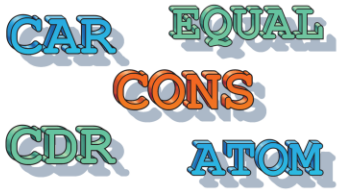
Добавление элемента к множеству

```
[1]> (setf *set* nil)
NIL
[2]> (setf *set* (adjoin "a" *set*
                         :test #'string=))
("a")
[3]> (setf *set* (adjoin "b" *set*
                         :test #'string=))
("b" "a")
[4]> (setf *set* (adjoin "a" *set*
                         :test #'string=))
("b" "a")
```



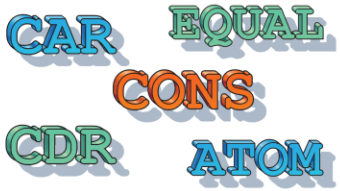
Проверка наличия элемента в множестве

```
[1]> (setf *set* '(1 2 3 4 5))
(1 2 3 4 5)

[2]> (member 3 *set*)
(3 4 5)

[3]> (member 10 *set*)
NIL
```



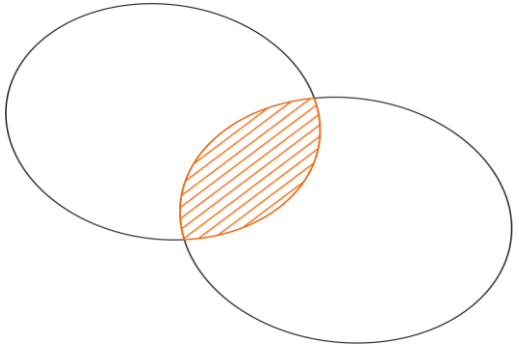
Проверка включения множеств

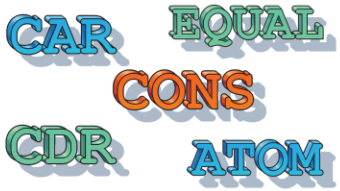
```
[1]> (subsetp      '(1 2 3)
                              '(1 2 3 4 5))
T
[2]> (subsetp      '(1 2 3)
                              '(2 3 4 5))
NIL
```



Пересечение множеств

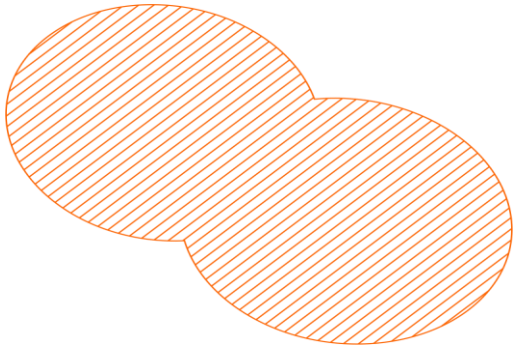
```
[1]> (intersection
                '(1 2 3)
                '(2 3 4))
(2 3)

[2]> (intersection
                '(1 2 3)
                '(11 22))
NIL
```

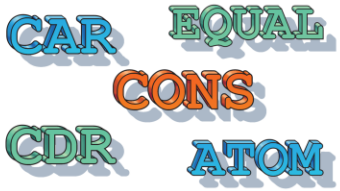




Объединение множеств

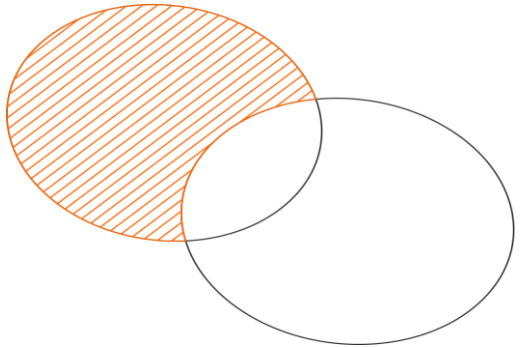
```
[1]> (union     '(1 2 3)
                '(2 3 4 5))
(1 2 3 4 5)
```

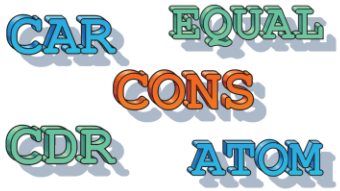




Разность множеств

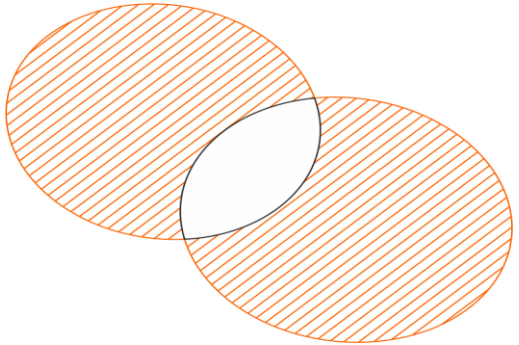
```
[1]> (set-difference
           '(1 2 3 4 5)
           '(1 3 5))
(2 4)

[2]> (set-difference
           '(1 2)
           '(5 6))
(1 2)
```

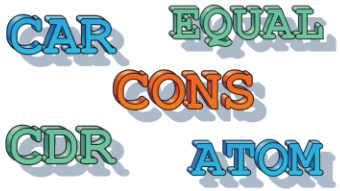




Симметрическая разность множеств

```
[1]> (set-exclusive-or
           '(1 2 3 4 5)
           '(2 4 6 8))
(1 3 5 6 8)

[2]> (set-exclusive-or
           '(1 2 3)
           '(2 3 1))
NIL
```





Что мы узнали из этой лекции

- наиболее часто используемый тип данных в Лиспе – списки, на их основе можно создавать различные структуры данных – множества, ассоциативные списки, стеки, деревья и т.д.
- для создания списков используются cons-ячейки
- Лисп также поддерживает динамические многомерные массивы, размер которых может изменяться по мере выполнения программы; и списки, и массивы в Лиспе могут включать элементы разных типов
- строки в Лиспе являются одномерными массивами символов