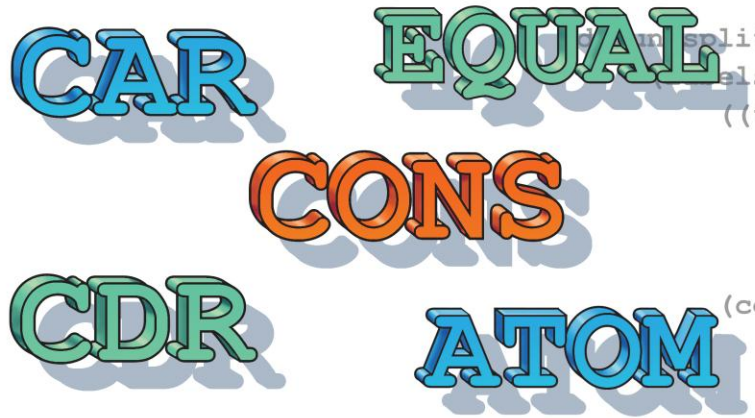
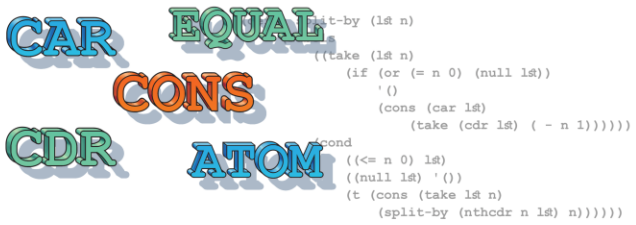
Функциональное программирование: базовый курс




```
(defun split-by (lst n)
  "docstring"
  ((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))))))
```

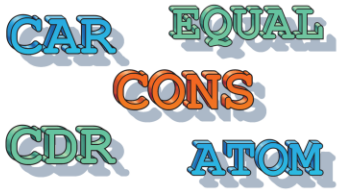
Гирик Алексей Валерьевич



Функциональное программирование: базовый курс

Лекция 1.

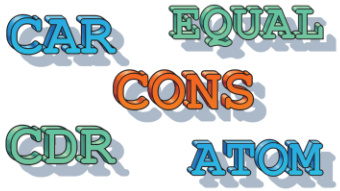
Введение в функциональное программирование.



Лекция 1

Введение в функциональное программирование

История развития языков программирования



История развития языков программирования

1834 — Чарльз Бэббидж начинает проектирование аналитической машины

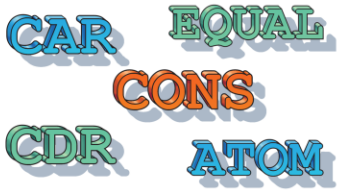
1843 — Ада Августа Лавлейс пишет первую в мире программу (алгоритм вычисления чисел Бернулли) для аналитической машины

1938 — Конрад Цузе в Германии создает первый в мире полностью программируемый компьютер Z1

1944 — Говард Айкен по контракту с IBM на основе идей Бэббиджа разрабатывает первый американский компьютер MARK I

1946 — Конрад Цузе разрабатывает первый в мире язык высокого уровня – Plankalkül (компилятор создан значительно позже – в 1975 году)

1948 — Сергей Алексеевич Лебедев начинает разработку первой в СССР ЭВМ – Малой Электронной Счетной Машины (МЭСМ)



Команда на языке ассемблера nasm для процессоров архитектуры x86

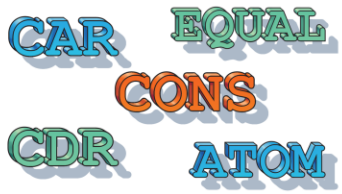
```
mov ecx, -1   ; записать -1 в регистр ecx
```

преобразуется в последовательность байтов (машинный код)

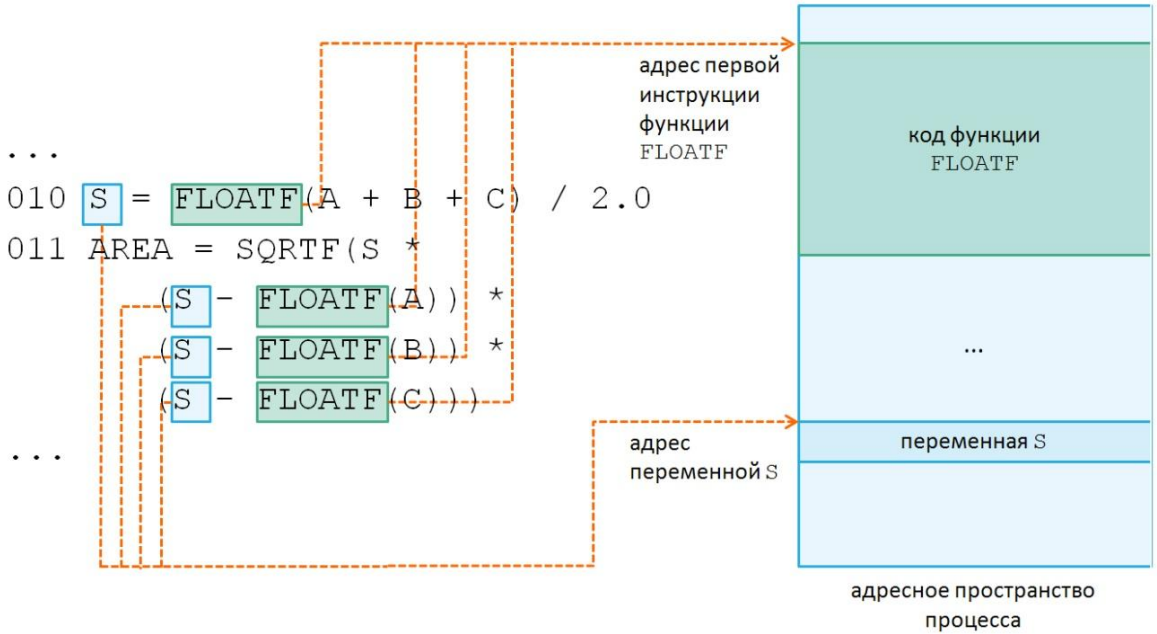
```
00 00 00 30 B9 FF FF FF FF
```

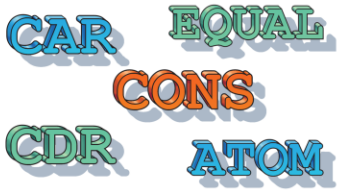
код операции (код команды, режим адресации и прочая информация)

–1



- 1957 год – FORTRAN, первый язык программирования высокого уровня





История развития языков программирования

1949 — Для компьютера EDSAC разработан первый ассемблер – программа преобразования мнемоник команд в машинный код

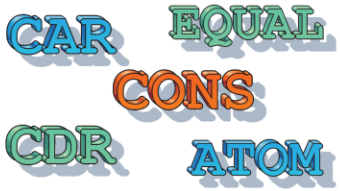
1954 — Сергей Камынин и Эдуард Любимский создают первый транслятор с языка высокого уровня ПП-1 (Программирующая программа 1)

1957 — Под руководством Джона Бэкуса создается FORTRAN – первый в мире язык высокого уровня, имеющий транслятор, который получил широкое признание

1958 — Джон Маккарти в MIT создает первый в мире язык для символьных вычислений – LISP

1973 — Кен Томпсон и Деннис Ритчи разрабатывают компилятор для языка С и переписывают на С ядро операционной системы UNIX

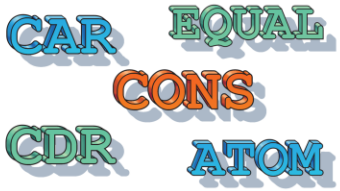
Ричард Гринблатт и Томас Найт в MIT создают первую ЛИСП-машину



Влияние естественных языков

Пример программы на языке AppleScript:

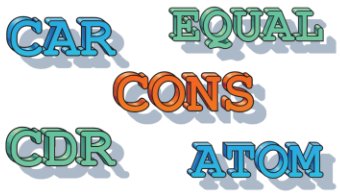
```
on adding folder items to F after receiving _list
     tell application "Finder"
          set c to 0
          repeat with _file in _list
               set n to name of _file
               if n ends with ".txt" then
                    tell application "Editor" to open _file
                    set c to c + 1
                    if c is greater than 100
                         tell application "Editor" to quit
                    end if
               end if
          end repeat
     end tell
end adding folder items to
```



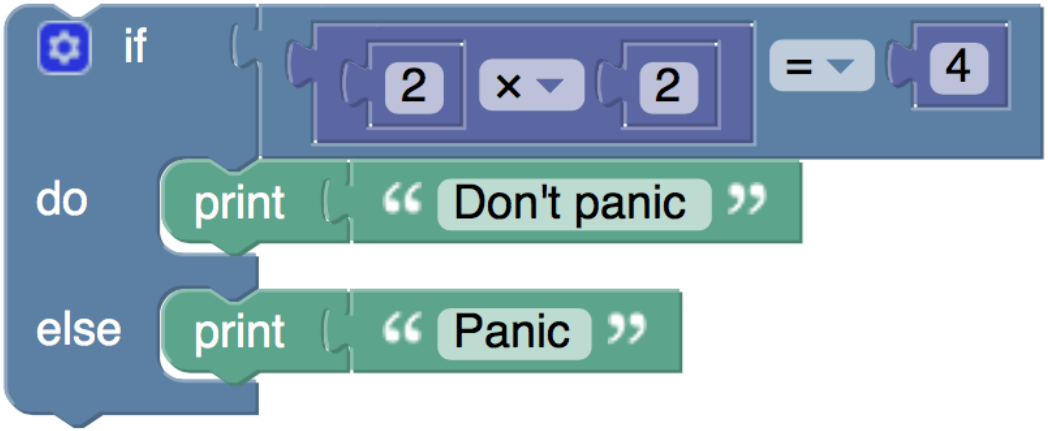
Влияние естественных языков

Пример программы на языке LOLCODE:

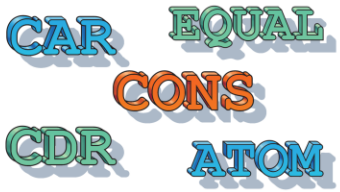
```
HAI
CAN HAS STDIO?
I HAS A VAR ITZ 0
VISIBLE NEED LOLZ HUH?
GIMMEH VAR              BTW LOLS COUNT
IM IN YR LOOP NERFIN YR VAR TIL BOTH SAEM VAR AN 0
     VISIBLE VAR :)
IM OUTTA YR LOOP
VISIBLE BYE! :0
KTHBYE
```



Пример кода на языке Blockly:

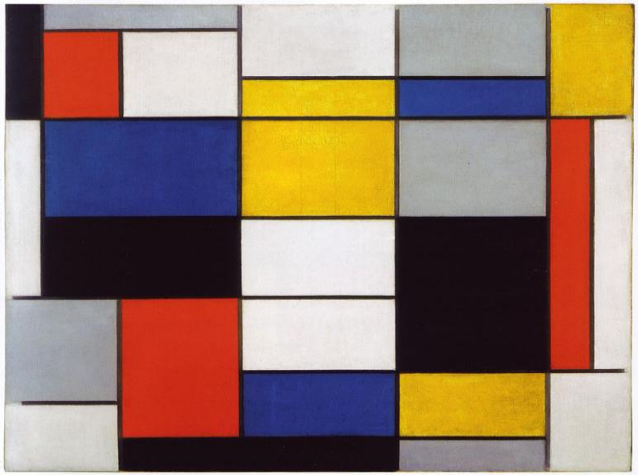


https://developers.google.com/blockly/

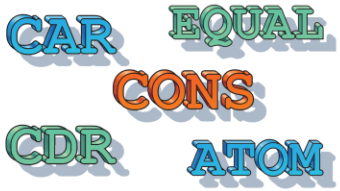


Необычные языки программирования

Piet,
программа вычисления
чисел Фибоначчи



Пит Мондриан,
"Композиция А", 1920



История развития языков программирования

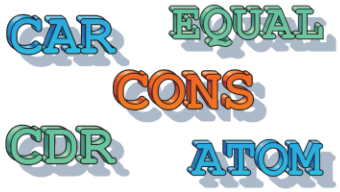
1975 — Гай Стил и Джеральд Сассман разрабатывают язык Scheme – минималистичный вариант ЛИСПа с поддержкой хвостовой рекурсии

1983 — Бьёрн Страуструп существенно дорабатывает С с классами и переименовывает язык в С++

1985 — Харольд Абельсон и Джеральд Сассман публикуют классический учебник по программированию "Structure and Interpretation of Computer Programs" с примерами на Scheme

1990 — Саймон Пейтон-Джонс, Филипп Уодлер и другие разрабатывают первый стандарт чистого функционального языка со строгой типизацией – Haskell

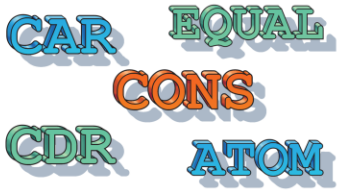
1995 — Группа под руководством Джеймса Гослинга из Sun Microsystems выпускает первую версию стандарта языка Java



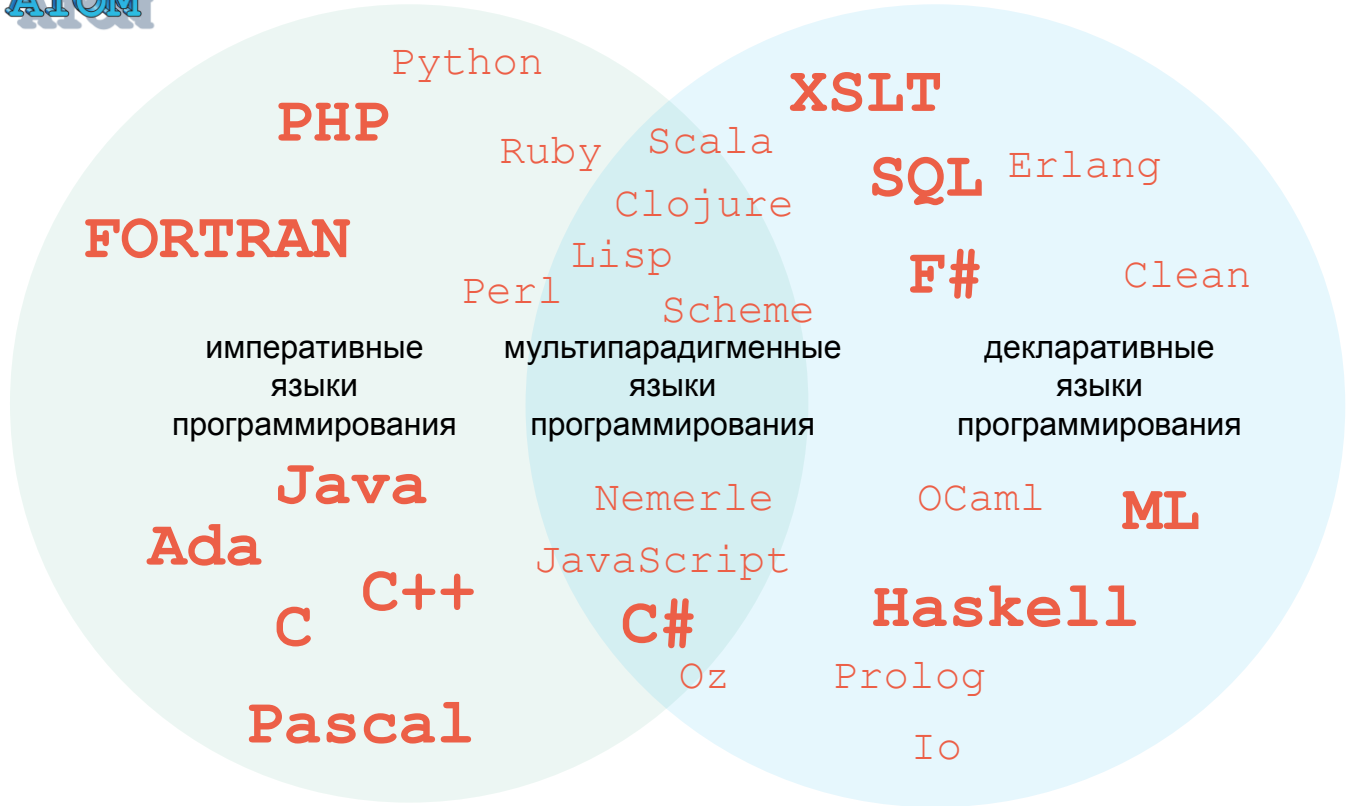
Лекция 1

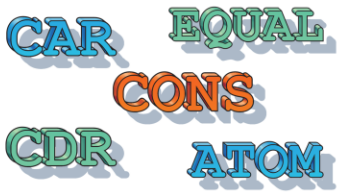
Введение в функциональное программирование

Парадигмы программирования



Парадигмы программирования и языки программирования





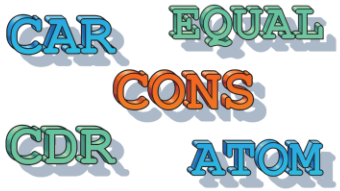
Императивная парадигма программирования

- исполнение программы – изменение состояния программы

$$S_0 \rightarrow S_1 \rightarrow \ldots \rightarrow S_N$$

```
mov [ecx], -1     ; ассемблер
k = -1;           // C
```

- явное управление порядком выполнения команд
  - условные операторы (if, switch и другие)
  - операторы циклов (while, for и другие)
  - операторы передачи управления (break, return, goto и другие)



Декларативная парадигма программирования

- исполнение программы – нахождение решения задачи по приведенной спецификации

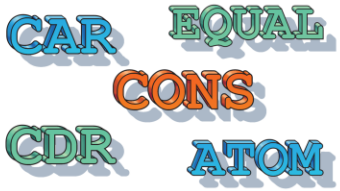
```
<?xml version="1.0" ?>
<persons>
  <person username="vasya">
    <name>Василий</name>
    <family-name>Пупкин</family-name>
  </person>
  <person username="jd">
    <name>John</name>
    <family-name>Doe</family-name>
  </person>
</persons>
```

пример определений на языке разметки XML

```
mother_child(lori, carl).
father_child(rick, carl).
father_child(rick, judith).
father_child(peter, rick).
sibling(X, Y)       :- parent_child(Z, X),
                       parent_child(Z, Y).
parent_child(X, Y) :- father_child(X, Y).
parent_child(X, Y) :- mother_child(X, Y).


?- sibling(carl, judith).
```

набор правил (отношений) на языке Prolog

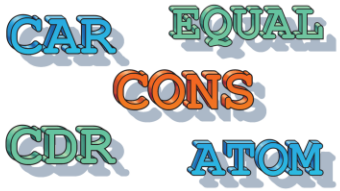
- отсутствие побочных эффектов (side effects), явных манипуляций с состоянием программы и детерминированного порядка выполнения



Функциональная парадигма программирования

$$f : X \to Y$$

- основные концепции функциональной парадигмы программирования:
  - функции высшего порядка (higher-order functions)
  - рекурсия (recursion)
  - чистые (pure) вычисления без побочных эффектов (side effects)
  - ленивые вычисления (lazy evaluation)



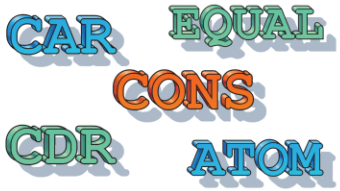
Сильные и слабые стороны функциональных языков

Достоинства

- большая выразительность синтаксиса и более плотная "упаковка" абстракций в текст программы
- модель вычислений без состояний упрощает тестирование программ и способствует получению более надежного кода
- более широкие возможности оптимизации по сравнению с императивными программами
- потенциальные возможности распараллеливания программ

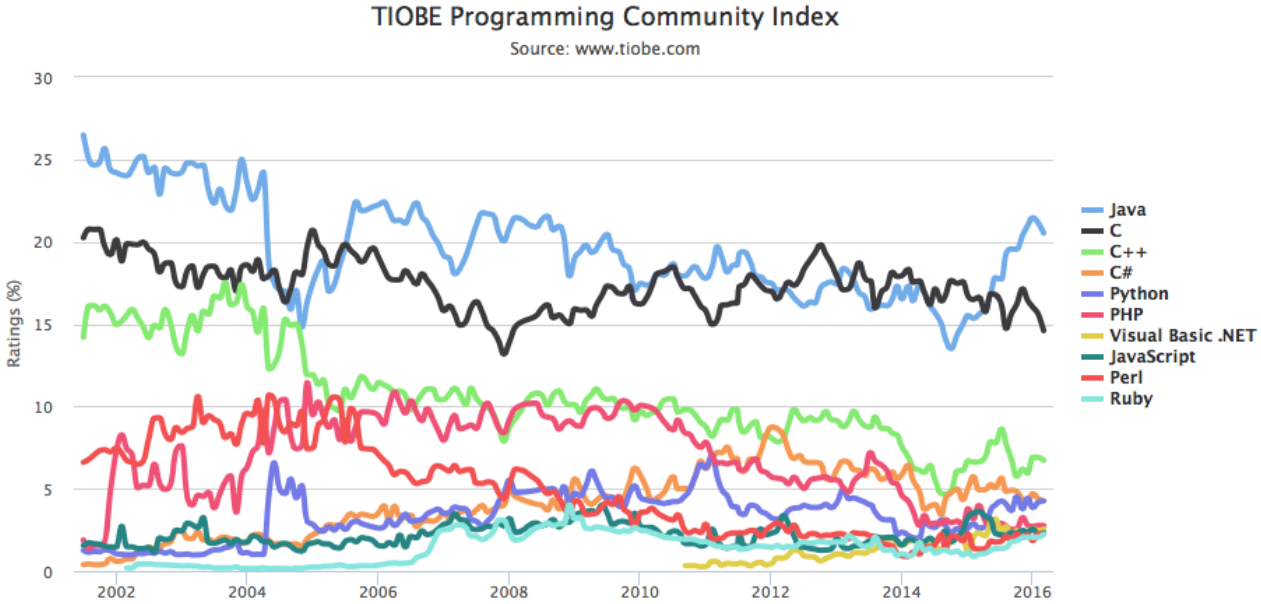
Недостатки

- более высокий "порог вхождения"
- более сложное управление памятью, чем в императивных языках
- потенциально большие накладные расходы и меньшее быстродействие по сравнению с императивными языками
- малая практичность чисто функциональных языков без императивных средств для решения "повседневных" задач

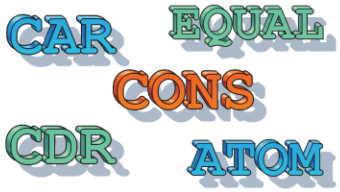


Популярность языков программирования

01. Java
02. C
…
28. **Lisp**
…
32. Prolog
…
34. Scheme
35. F#
…
38. Haskell
…
40. Erlang
…
64. Clojure
…



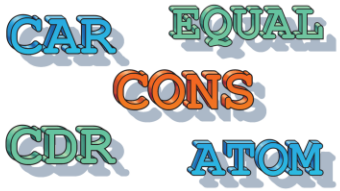
http://www.tiobe.com/tiobe_index



Лекция 1

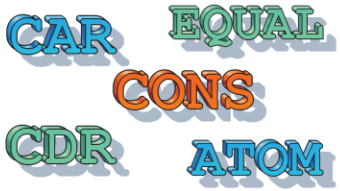
Введение в функциональное программирование

Семейство языков Lisp

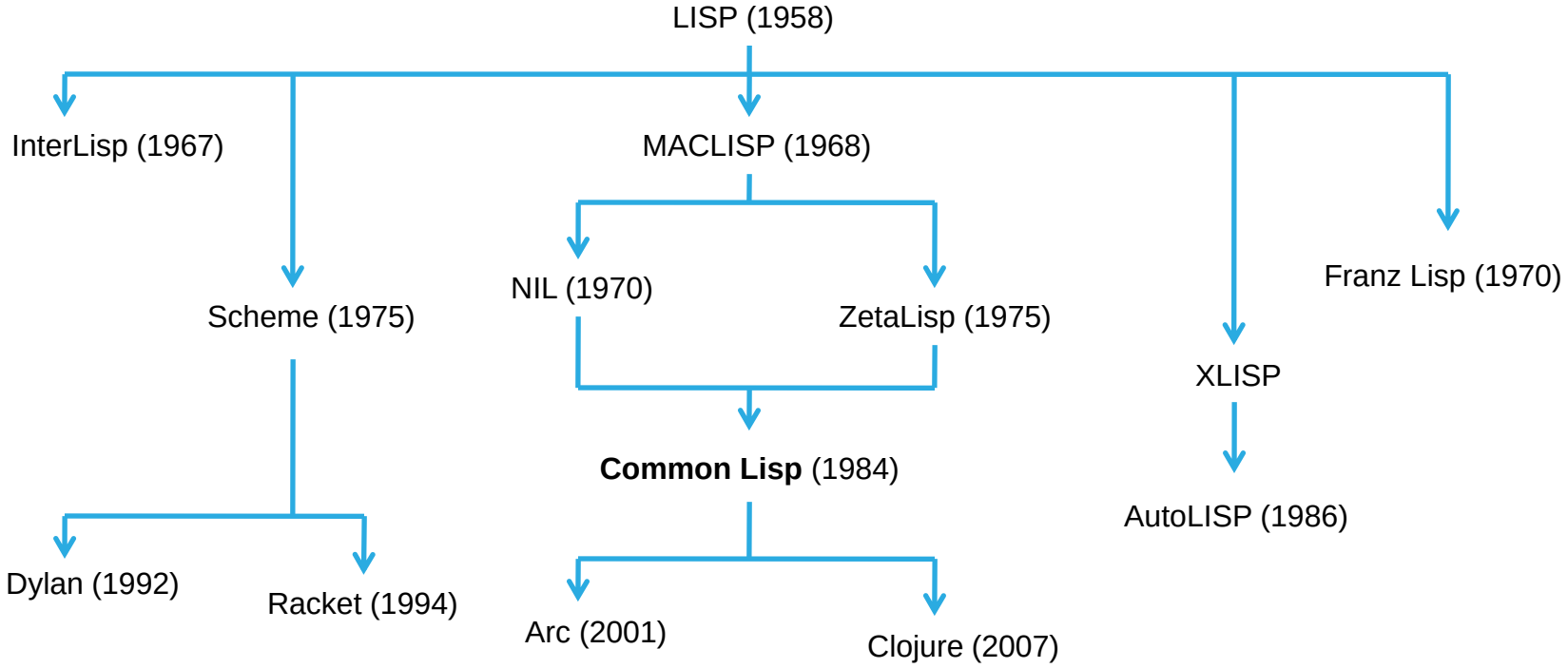


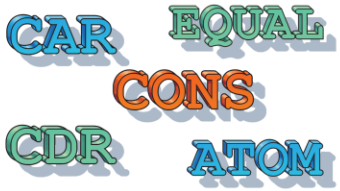
Почему Lisp?

- простой синтаксис, легко допускающий расширение
- поддержка нескольких парадигм
- большое количество качественных библиотек
- крупное сообщество разработчиков



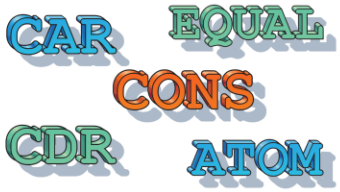
Семейство языков Lisp





Наиболее популярные реализации Common Lisp

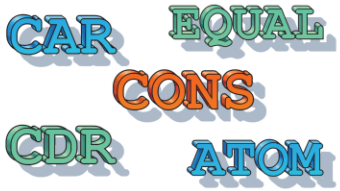
- **CLISP**
  http://www.clisp.org/
- Steel Bank Common Lisp
  http://www.sbcl.org
- Allegro Common Lisp
  http://franz.com/products/allegro-common-lisp/
- LispWorks
  http://www.lispworks.com/



Лекция 1

Введение в функциональное программирование

Основные сведения об исполнении программ на Лиспе и синтаксисе языка



Трансляторы, интерпретаторы и компиляторы

```
...
create_file("c:\my_file.txt");

write_file("c:\my_file.txt",
 "Don't trouble trouble
  until trouble troubles you");

save_file("c:\my_file.txt");
...
```

файл с исходным текстом

0010010100101 … 00101

1101001010101 … 001010
0010010010011101010101

01110101110010 … 11100

интерпретатор

создание файла

запись текста в файл

сохранение файла

```
...
create_file("c:\my_file.txt");

write_file("c:\my_file.txt",
 "Don't trouble trouble
  until trouble troubles you");

save_file("c:\my_file.txt");
...
```

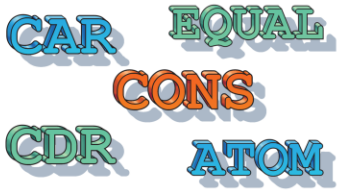
файл с исходным текстом

11011101010010101010101
00010000010001010001010
10101001111010111101101
…
10001010100010100010101

компилятор

сохранение на диске
в виде исполняемого
модуля

My super app.exe



Компиляция в промежуточное представление

```
...
create_file("c:\my_file.txt");

write_file("c:\my_file.txt",
 "Don't trouble trouble
  until trouble troubles you");

save_file("c:\my_file.txt");
...
```

файл с исходным текстом

компилятор

```
11011101010010101010
00010000010001010010
10101001111011110110
...
10001010100010100010
1
```

файл с промежуточным представлением программы (байт-код)

```
11011101010010101010
00010000010001010010
10101001111011110110
...
10001010100010100010
1
```

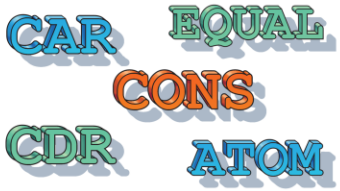
среда исполнения/ виртуальная машина

выполнение действий программы

создание файла

запись текста в файл

сохранение файла



Исполнение программ на Лиспе

1

Написание программы и сохранение текста в файле (обычно файлам с программами на Лиспе дают расширение .lisp или .lsp)

2a

Исполнение программы как сценария путем передачи файла с исходным кодом интерпретатору

2b

Компиляция в промежуточное представление (байт-код). Файлы с байт-кодом обычно имеют расширение .fas или .fasl

3a

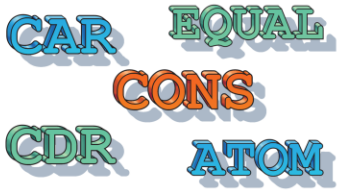
Исполнение программы путем запуска среды исполнения и передачи ей файла с байт-кодом

2c

Компиляция в машинный код и сохранение в виде исполняемого файла

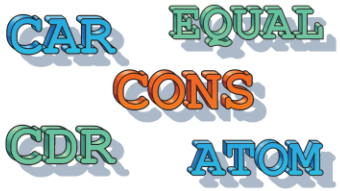
3b

Запуск исполняемого модуля



Read-Eval-Print Loop

```
$ clisp

[1]> 42
42
[2]> (40 + 2)

*** - EVAL: 40 is not a function name; try using a symbol
The following restarts are available:
USE-VALUE      :R1      Input a value to be used instead.
ABORT          :R2      Abort main loop

Break 1[3]> :r2

[4]> (+ 40 2)
42
[5]> (quit)
Bye.

$
```



s-выражения и формы

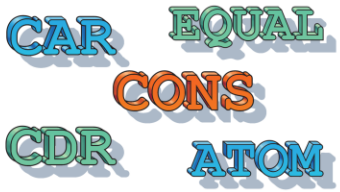
- s-выражения (s-expressions) – основа синтаксиса Лиспа. s-выражение может быть либо отдельным атомом, либо списком, содержащим другие s-выражения:

```
(a1 a2 a3 … aN)
```

- формы – s-выражения, которые предназначены для исполнения Лиспом:

```
(operation arg1 arg2 … argN)
```

```
[1]> (+ 1 2 3)
6
[2]> (+ 1 2)
3
[3]> (+ 1)
1
[4]> (+)
0
[5]> ()
NIL
```



Порядок записи выражений в формах

40 + 2

инфиксная нотация

40 2 +

постфиксная нотация

+ 40 2

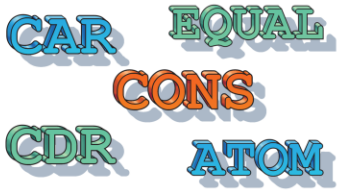
префиксная нотация

2 + 2 == 4

так выражение проверки на равенство будет записано в C/C++

(= (+ 2 2) 4)

так то же самое выражение будет записано в Лиспе



Первая программа

- Создадим в каталоге /test файл hello.lisp со следующим содержимым:

```
(print "Hello, Lisp!")
```

- Запустим интерпретатор и загрузим файл для исполнения:

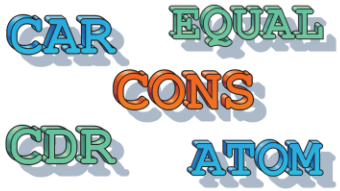
```
[1]> (load "/test/hello.lisp")
"Hello, Lisp!"
```

- Можем скомпилировать hello.lisp в промежуточное представление:

```
[2]> (compile-file "/test/hello.lisp")
```

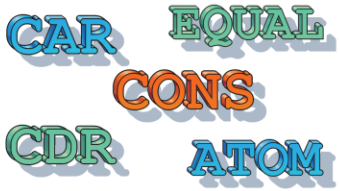
- Позднее можем загрузить скомпилированную программу:

```
[3]> (load "/test/hello.fas")
"Hello, Lisp!"
```



Первая программа

```
$ clisp /test/hello.lisp
"Hello, Lisp!"

$ clisp -c /test/hello.lisp
;; Compiling file /test/hello.lisp ...
;; Wrote file /test/hello.fas
0 errors, 0 warnings
Bye.

$ clisp /test/hello.fas
"Hello, Lisp!"
```



Что мы узнали на этой лекции

- какова краткая история возникновения языков программирования
- что такое парадигмы программирования, как они развивались и каково современное состояние
- каковы сильные и слабые стороны императивной и функциональной парадигм
- какие языки входят в семейство языков Lisp
- что такое трансляция, компиляция и интерпретация программ
- как исполняются программы на Лиспе
- что такое Read-Eval-Print loop, s-выражения и формы