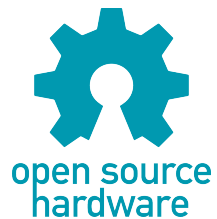




# Скрипт-язык bl

© Дмитрий Понятов <[dponyaton@gmail.com](mailto:dponyaton@gmail.com)>



# Оглавление

|  |          |
|--|----------|
| Цели . . . . .                                 | 1        |
| Установка . . . . .                            | 1        |
| <b>1 Учебник</b>                               | <b>3</b> |
| Запуск . . . . .                               | 3        |
| 1.1 Базовый AST-тип . . . . .                  | 4        |
| 1.2 Скалярные типы (атомы, литералы) . . . . . | 5        |
| 1.2.1 Комментарии . . . . .                    | 5        |
| 1.2.2 docstring: строка документации . . . . . | 5        |
| 1.2.3 Числа, строки и символы . . . . .        | 6        |
| 1.3 Композитные типы . . . . .                 | 6        |
| 1.3.1 [Список] . . . . .                       | 6        |
| 1.3.2 Па:ра . . . . .                          | 7        |
| 1.3.3 <Вектор> . . . . .                       | 7        |
| 1.3.4 [Сло:варь] . . . . .                     | 8        |
| 1.4 Заимствования из языка Forth . . . . .     | 8        |
| 1.4.1 Стек данных . . . . .                    | 8        |
| 1.4.2 Forth-функции на стеке данных . . . . .  | 10       |

|          |   |           |
|----------|---|-----------|
| 1.5      | Функциональные типы . . . . .                 | 11        |
| 1.5.1    | Лямбды . . . . .                              | 11        |
| 1.5.2    | Определение именованных функций . . . . .     | 11        |
| 1.5.3    | Явная аппликация . . . . .                    | 12        |
| 1.5.4    | Декомпозиция: программы как данные . . . . .  | 12        |
| 1.6      | ООП . . . . .                                 | 12        |
| <b>2</b> | <b>Библиотеки</b>                             | <b>13</b> |
| 2.1      | Математика . . . . .                          | 14        |
| 2.1.1    | Константы . . . . .                           | 14        |
| 2.1.2    | Стандартные функции . . . . .                 | 14        |
| 2.1.3    | Тригонометрия . . . . .                       | 15        |
| 2.2      | Операции с файлами и каталогами . . . . .     | 16        |
| 2.2.1    | Файловые классы . . . . .                     | 16        |
| 2.2.2    | Типовые операции командной оболочки . . . . . | 16        |
| <b>3</b> | <b>Синтаксис</b>                              | <b>17</b> |
| <b>4</b> | <b>Реализация</b>                             | <b>18</b> |
| <b>5</b> | <b>Мета-модель</b>                            | <b>19</b> |
|          | <b>Литература</b>                             | <b>19</b> |

# Цели

- универсальный язык описания входных данных для расчетных программ и файлов конфигурации
- средство программирования пользовательского GUI (реализация APM)
- высокоуровневый экспериментальный язык-шаблонизатор для разработки средств трансляции, компиляторов, синтаксического анализа текстовых форматов данных, автогенерации кода на базе высокоуровневых определений, и символьных вычислений.
  - генерация проектов по набору шаблонных файлов
  - мульти-платформенная разработка ПО
  - формирование файлов конфигурации и управления работой кластерного ПО
  - анализ и преобразование текстовых форматов данных и программ

## Установка

- GitHub: <https://github.com/ponyatov/script.git>
- Ручная загрузка:
  - win32 .exe  
<https://github.com/ponyatov/script/raw/master/bI.exe>
  - руководство .pdf  
[https://github.com/ponyatov/bI\\_manual/raw/master/manual.pdf](https://github.com/ponyatov/bI_manual/raw/master/manual.pdf)
  - тестовый bI-скрипт  
<https://github.com/ponyatov/script/raw/master/bI.bI>

- Сборка из исходных файлов:

- Windows (win32):

- \* пакет компилятора GNU GCC/G++

- MinGW <http://www.mingw.org/download/installer?>

- \* Система управления версиями Git

- [git-scm https://git-scm.com/downloads](https://git-scm.com/downloads)

- \* Текстовый редактор Vim

- [gvim ftp://ftp.vim.org/pub/vim/pc/gvim74.exe](ftp://ftp.vim.org/pub/vim/pc/gvim74.exe)

- ```
git clone -o gh https://github.com/ponyatov/script.git bI_script
```

- ```
cd bI_script
```

- ```
mingw32-make EXE=.exe RES=res.res
```

- Linux:

- ```
git clone -o gh https://github.com/ponyatov/script.git bI_script
```

- ```
cd bI_script
```

- ```
make EXE=.elf RES=
```

# Глава 1

## Учебник

### Запуск

Ядро системы реализовано в виде единственного .exe файла<sup>1</sup>. При простом запуске открывается интерактивная сессия, для загрузки скрипта из файла используйте команду

```
bI.exe < script.bI > script.blog
```

В Linux исполняемым может быть сделан *любой* файл, добавьте к своим скриптам первой строкой полный путь к bl.elf:

```
#!/home/user/bI_script/bI.elf
```

и выполните команду

```
chmod +x script.bI
```

---

<sup>1</sup> консольная программа

## 1.1 Базовый AST-тип

Язык bl реализован на базе операций с элементами данных, представленных в виде **символьного типа** AST:

```
class AST:sym
    string:tag          тип данных (тэг класса)
    string:value         значение
    sym:nest[]          список вложенных элементов данных
    sym:par{}           словарь параметров
    string:dump()        вывод элемента данных в текстовом виде в виде дерева
    string:tagval()      строковое представление основной части <тэг:значение[,параметры]>
    sym:eval()           вычисление (evaluate) элемента данных
```

Базовый объект задается **парой** 1.3.2 тэг:значение.

Каждый AST-объект может иметь вложенные элементы в списке next [].

Также каждому элементу можно присваивать произвольное количество параметров, хранящихся в словаре<sup>2</sup> par{ }.

Относительно сложный AST-тип (по сравнению с списковыми типами Lisp) был выбран в качестве базового, так как изначально язык bl создавался как *высокоуровневый экспериментальный язык-шаблонизатор* для разработки средств трансляции, компиляторов, синтаксического анализа текстовых форматов данных, автогенерации кода на базе высокоуровневых определений, и символьных вычислений.

---

<sup>2</sup> тип Python

## 1.2 Скалярные типы (атомы, литералы)

### 1.2.1 Комментарии

Строчные комментарии начинаются с символа `#`:

Создайте файл `script.bl`, запишите в него

```
#!/home/user/bI_script/bI.elf
# syntax sample with numbers, symbols, [] lists and stack ops
# John McCarthy A Micro-Manual for Lisp - not the whole Truth.
```

и выполните через `bl`-интерпретатор:

```
bI.exe < script.bl
```

### 1.2.2 `docstring`: строка документации

Для документирования программ из языка Python была позаимствована идея `docstring`: каждый объект может иметь параметр `par[doc]`, в котором хранится объект документации: строка или более сложный тип.

Для задания `docstring` в тексте программ используются "двойные кавычки": созданная докстрока будет присвоена *ближайшему слева* программному объекту<sup>3</sup>.

---

<sup>3</sup> правая ассоциативность



## 1.2.3 Числа, строки и символы

- числа
  - целые:  
`-01 -0 00 +0 +02222` → `<int:-1>` `<int:0>` `<int:0>` `<int:0>` `<int:2222>`
  - с плавающей точкой в простой и экспоненциальной форме:  
`+02.30 -04E+05` → `<num:2.3>` `<num:-400000>`
  - шестнадцатеричные и бинарные машинные числа:  
`0x12AF 0b1101` → `<hex:0x12AF>` `<bin:0b1101>`
- строки  
`'строка'` → `<str:'строка'>`
- символы: в простейшем виде просто задает уникальное имя  
`symbol` → `<sym:symbol>`

## 1.3 Композитные типы

### 1.3.1 [Список]

Список — **композитная** структура данных, позволяющая рассматривать произвольный набор данных как один объект. Списки могут быть вложенными.

Добавьте в `script.bl` или выполните в интерактивной сессии<sup>4</sup> код:

```
# пример синтаксиса с числами, символами, и вложенными списками:
```

```
[
```

---

<sup>4</sup> запустив `bl.exe` без параметров

```
[-01 -0 00 +0 +02222] "целые"  
[ 0x12AF 0b1101 ] "машинные числа"  
[ +02.30 -04E+05 ] "числа с плавающей точкой"  
]  
'string' "пример строки"  
symbol "пример символа"  
? # вывести дамп стека  
~ # dropall: полностью очистить стек данных  
? # еще раз вывести стек
```

## 1.3.2 Па:ра

Через **пары** вида А:В реализуются деревья, списки в Lisp-стиле, и описание ООП-структур.

## 1.3.3 <Вектор>

<Вектор> аналогичен списку, но имеет *фиксированную размерность*.

Фиксированная размерность позволяет генерировать оптимизированный код, и значительно упрощает управление памятью, что критически важно для реализации численных методов.

1.3.4 [Сло:варь]

1.4 Заимствования из языка Forth

В языке Forth программа представляет собой последовательность слов-команд, разделенных пробелами<sup>5</sup>. Когда интерпретатор Forth встречается слово, он ищет его в словаре, и если оно найдено — исполняет.

Если слово не найдено, Forth пытается его прочесть как целое число, при успехе кладет на стек, иначе выводит ошибку. Плавающей точки нет вообще.

Строки обрабатываются особо: слово ." <sup>6</sup> читает входной поток, пока не встретит символ ", и кладет на стек адрес строки.

Язык bl расширяет синтаксис Forth<sup>7</sup> инфиксными выражениями, встроенными типами данных, высоко-уровневой библиотекой символьных вычислений и ООП.

1.4.1 Стек данных

Каждое выражение языка bl после вычисления кладется на стек данных.

Стек — структура данных, поддерживающая операции:

|                       |                       |                                   |
|-----------------------|-----------------------|-----------------------------------|
| push(объект)          | ( A B C – A B C obj ) | поместить объект на вершину стека |
| pop()→объект          | ( A B C obj – A B C ) | взять объект с вершины            |
| стековые функции1.4.2 |                       | выполняют вычисления на стеке     |

<sup>5</sup> сам пробел, табуляция и концы строк DOS/UNIX

<sup>6</sup> с обязательным за ним пробелом

<sup>7</sup> рассматривая все данные как обобщенный AST-тип1.1

Для записи состояния стека используется строчная Forth-нотация: элементы записываются слева направо, вершина — самый правый элемент. Для встроенных Forth-функций 1.4.2 используется нотация вида `A B C -- A C B`, в левой части состояние стека до выполнения функции, справа — после того, как функция выполнила **вычисления на стеке данных**.

? вывести стек (вершина внизу)  
~ очистить стек

Команда ? выводит стек в полной текстовой форме в виде дерева, для составных объектов включаются все вложенные элементы. Стек выводится сверху вниз, вершина стека — самый нижний элемент.

При выполнении примера 1.3.1 будет выведено (список списков чисел, строка и символ):

```
===== stack =====
<[:]>
  <[:],doc:<str:'integers'>>
    <int:-1>
    <int:0>
    <int:0>
    <int:0>
    <int:2222>
  <[:],doc:<str:'machine numbers'>>
    <hex:0x12AF>
    <bin:0b1101>
  <[:],doc:<str:'float numerics'>>
    <num:2.3>
    <num:-400000>
<str:'string'>
```

```
<sym:symbol,doc:<str:'generic symbol'>>
```

```
=====
```

```
===== stack =====
```

```
=====
```

0ой элемент = вершина стека — символ;

-1ый элемент строка;

-2ой элемент = дно стека — вложенный список чисел разных типов.

## 1.4.2 Forth-функции на стеке данных

Стек данных и Forth-функции были введены в язык bl как упрощение, позволяющее реализовать лямбды и определение функций.

Явное использование стека данных позволяет определять лямбды как простой список имен ранее определенных и встроенных функций, выполняющих операции на стеке.

Это позволяет обойтись без подстановки параметров в традиционном стиле реализации лямбда-функций.

Forth-функции выполняют вычисления на стеке данных, явно манипулируя элементами на вершине:

|              |                       |  |
|--------------|-----------------------|--|
| push(объект) | ( A B C – A B C obj ) | поместить объект на вершину стека            |
| pop()→объект | ( A B C obj – A B C ) | взять объект с вершины                       |
| ?            | аналог .S             | вывести полный дамп стека                    |
| ~            | аналог DROPALL        | очистить стек                                |
| print        | аналог . ( A B – A )  | вычислить и вывести символьное представление |
| dup          | ( A B C – A B C C )   | продублировать верхний элемент               |
| drop         | ( A B C – A B )       | убрать один элемент с вершины                |
| swap         | ( A B C – A C B )     | обмен 2х объектов на стеке                   |
| over         | ( A B C – A B C B )   | вытащить копию -1-ого элемента на вершину    |

## 1.5 Функциональные типы

### 1.5.1 Лямбды

Лямбда — анонимная функция. В текущей версии bl классические лямбды с параметрами не поддерживаются, доступно только определение в виде Forth-функций 1.4.2, выполняющих вычисления на стеке данных. Эта реализация намного проще в реализации, но имеет значительный недостаток — *вычисления выполняются декструктивно, а не функционально*.

### 1.5.2 Определение именованных функций

Для задания именованной функции достаточно присвоить лямбду символу:

$\# x \rightarrow \sin(x) \rightarrow \sin(x) \sin(x) \rightarrow \sin^2(x)$

sin2x = { sin apply dup apply mul apply }

### 1.5.3 Явная аппликация

Поскольку в `bl` функции являются таким же типом данных, как и остальные, для применения (аппликации) функции, т.е. вычисления функции, применяется встроенная функция `apply` и оператор `@`.

### 1.5.4 Декомпозиция: программы как данные

Важнейшее значение в языке `bl` имеет заимствованный из `Lispa` принцип "программа = данные" и поддержка ядром языка функций высших порядков:

**Функция высшего порядка** — функция, способная создавать, принимать в качестве параметров, и модифицировать другие функции.

Собственно, язык `bl` и был создан из необходимости наличия в языке ФВП, полного набора возможностей динамического языка, и с удобным (и более вменяемым по сравнению с `Lispa`) синтаксисом.

## 1.6 ООП

## Глава 2

### Библиотеки



# 2.1 Математика

## 2.1.1 Константы

| символьная | численная |                                  |              |
|------------|-----------|----------------------------------|--------------|
| E          | e         | основание натурального логарифма | $e$          |
| PI         | pi        | число                            | $\pi$        |
|            | c         | скорость света в вакууме         | $c$          |
|            | h         | постоянная Планка                | $h$          |
|            | G         | гравитационная постоянная        | $G$          |
|            | g         | ускорение свободного падения     | $g$          |
|            | e         | элементарный электрический заряд | $e$          |
|            | k         | постоянная Больцмана             | $k$          |
|            | Na        | постоянная Авогадро              | $N_a$        |
| E0         | e0        | (ди)электрическая постоянная     | $\epsilon_0$ |
| U0         | u0        | магнитная постоянная             | $\mu_0$      |

## 2.1.2 Стандартные функции

|       |                                 |                      |
|-------|---------------------------------|----------------------|
| sqrt  | $x \rightarrow \sqrt{x}$        | квадратный корень    |
| sqrn  | $x \ n \rightarrow \sqrt[n]{x}$ |                      |
| exp   | $x \rightarrow e^x$             |                      |
| pow   | $x \ y \rightarrow x^y$         |                      |
| ln    | $x \rightarrow \ln x$           | натуральный логарифм |
| log10 | $x \rightarrow \log_{10} x$     | десятичный логарифм  |

### 2.1.3 Тригонометрия

|                       |  |           |
|-----------------------|--|-----------|
| $\sin$                | $x \rightarrow \sin(x)$                  | синус     |
| $\cos$                | $x \rightarrow \cos(x)$                  | косинус   |
| $\tan$                | $x \rightarrow \tan(x)$                  | тангенс   |
| $\operatorname{ctg}$  | $x \rightarrow \operatorname{ctg}(x)$    | котангенс |
| $\operatorname{asin}$ | $x \rightarrow \arcsin(x)$               | арксинус  |
| $\operatorname{acos}$ | $x \rightarrow \arccos(x)$               |           |
| $\operatorname{atan}$ | $x \rightarrow \operatorname{atctan}(x)$ |           |

## 2.2 Операции с файлами и каталогами

### 2.2.1 Файловые классы

```
class:fileio  
fileio:dir  
fileio:file
```

### 2.2.2 Типовые операции командной оболочки

`mkdir str → dir` создать каталог

# Глава 3

## Синтаксис

## Глава 4

# Реализация

# Глава 5

## Мета-модель

# Литература

- [1] A micro-manual for LISP Implemented in C  
<http://nakkaya.com/2010/08/24/a-micro-manual-for-lisp-implemented-in-c/>
- [2] John McCarthy  
A Micro-Manual for Lisp — not the whole Truth