

Лабораторная работа № 1. Julia. Установка и настройка. Основные принципы.

Абакумова Олеся Максимовна, НФИбд-02-22

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
3.1	Подготовка инструментария к работе	7
3.2	Основы работы в блокноте Jupyter	9
3.3	Основы синтаксиса Julia на примерах	12
3.4	Задания для самостоятельной работы	16
4	Выводы	23

Список иллюстраций

3.1	Запуск Julia	7
3.2	Запуск Jupyter	7
3.3	Запуск Jupyter	8
3.4	Создание нового блокнота для Julia	8
3.5	Блокнот имеет ядро Julia	9
3.6	Режимы вставки ячейки	9
3.7	Простейшие операции на языке Julia в Jupyter Lab	10
3.8	Пример получения информации по функции println на языке Julia в Jupyter Lab	11
3.9	Пример получения информации о дате и пользователе ОС Linux в Jupyter Lab	11
3.10	Очистка результатов выполнения ячеек	12
3.11	Определение крайних значений диапазонов целочисленных числовых величин и определение типа числовой величины	13
3.12	Примеры приведения аргументов к одному типу	14
3.13	Пример определения одномерных массивов и пример определения функций	15
3.14	Пример определения функций	15
3.15	Примеры работы с массивами	16
3.16	Функции read(), readline(), readlines()	17
3.17	Функции readlm(), print(), println(), show(), write()	18
3.18	Функция parse()	18
3.19	Математические операции с разным типом переменных	20
3.20	Математические операции с разным типом переменных(логические операции)	21
3.21	Операции над матрицами и векторами	22

Список таблиц

1 Цель работы

Основная цель работы – подготовить рабочее пространство и инструментарий для работы с языком программирования Julia, на простейших примерах познакомиться с основами синтаксиса Julia.

2 Задание

1. Установите под свою операционную систему Julia, Jupyter.
2. Используя Jupyter Lab, повторите примеры из разделов.
3. Выполните задания для самостоятельной работы.

3 Выполнение лабораторной работы

3.1 Подготовка инструментария к работе

Для выполнения данной лабораторной работы нам необходимы Julia и Jupyter. Так, как они установлены у меня уже изначально, то могу продемонстрировать их наличие на моей ОС (рис. 3.1):

```
lesya@lesya-Aspire-A515-32:~$ julia
The latest version of Julia in the 'release' channel is 1.11.6+0.x64.linux.gnu. You currently have '1.11.3+0.x64.linux.gnu' installed. Run:
juliaup update
In your terminal shell to install Julia 1.11.6+0.x64.linux.gnu and update the 'release' channel to that version.
Documentation: https://docs.julialang.org
Type "?" for help, "j?" for pkg help.
Version 1.11.3 (2025-01-21)
Official https://julialang.org/ release
julia>
```

Рис. 3.1: Запуск Julia

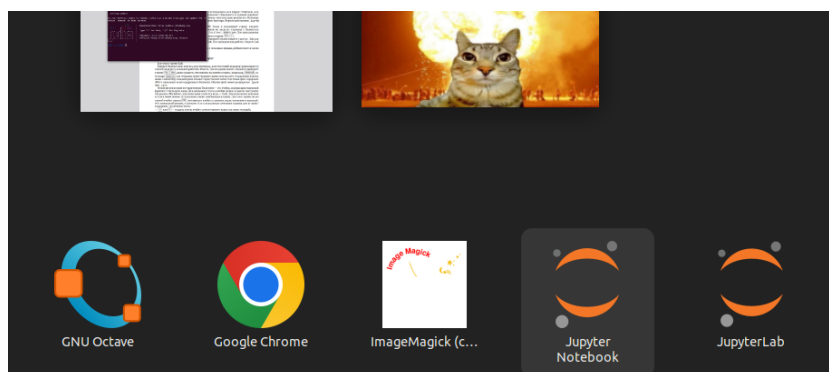
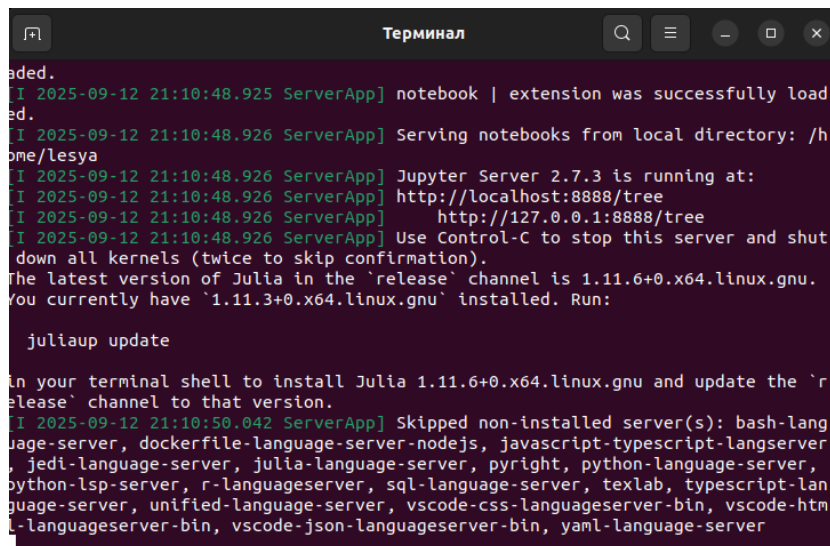


Рис. 3.2: Запуск Jupyter



```
added.
[I 2025-09-12 21:10:48.925 ServerApp] notebook | extension was successfully load
ed.
[I 2025-09-12 21:10:48.926 ServerApp] Serving notebooks from local directory: /h
ome/lesya
[I 2025-09-12 21:10:48.926 ServerApp] Jupyter Server 2.7.3 is running at:
[I 2025-09-12 21:10:48.926 ServerApp] http://localhost:8888/tree
[I 2025-09-12 21:10:48.926 ServerApp] http://127.0.0.1:8888/tree
[I 2025-09-12 21:10:48.926 ServerApp] Use Control-C to stop this server and shut
down all kernels (twice to skip confirmation).
The latest version of Julia in the 'release' channel is 1.11.6+0.x64.linux.gnu.
You currently have '1.11.3+0.x64.linux.gnu' installed. Run:

juliaup update

In your terminal shell to install Julia 1.11.6+0.x64.linux.gnu and update the 'r
elease' channel to that version.
[I 2025-09-12 21:10:50.042 ServerApp] Skipped non-installed server(s): bash-lang
uage-server, dockerfile-language-server-nodejs, javascript-typescript-langserver
, jedi-language-server, julia-language-server, pyright, python-language-server,
python-lsp-server, r-languageserver, sql-language-server, texlab, typescript-lan
guage-server, unified-language-server, vscode-css-languageserver-bin, vscode-htm
l-languageserver-bin, vscode-json-languageserver-bin, yaml-language-server
```

Рис. 3.3: Запуск Jupyter

Зайдя в Jupyter Notebook, создадим блокнот для работы с Julia (рис. 3.4):

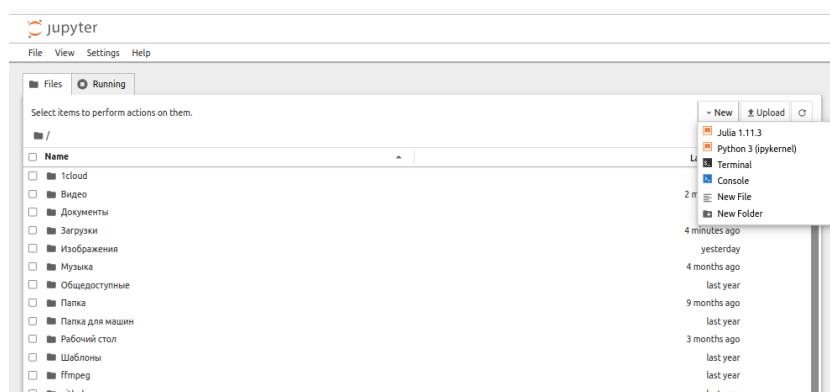


Рис. 3.4: Создание нового блокнота для Julia

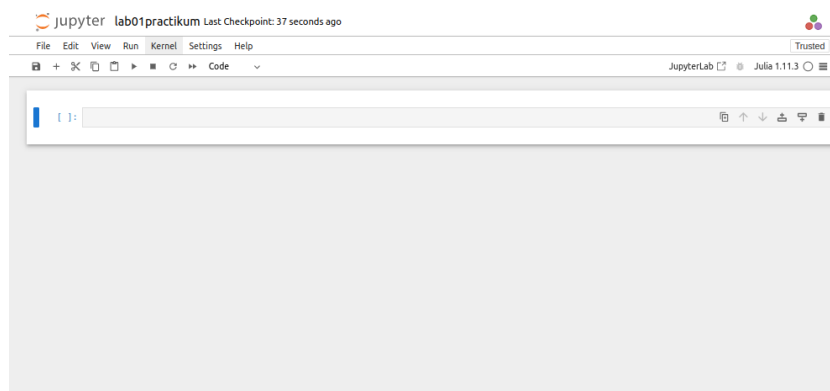


Рис. 3.5: Блокнот имеет ядро Julia

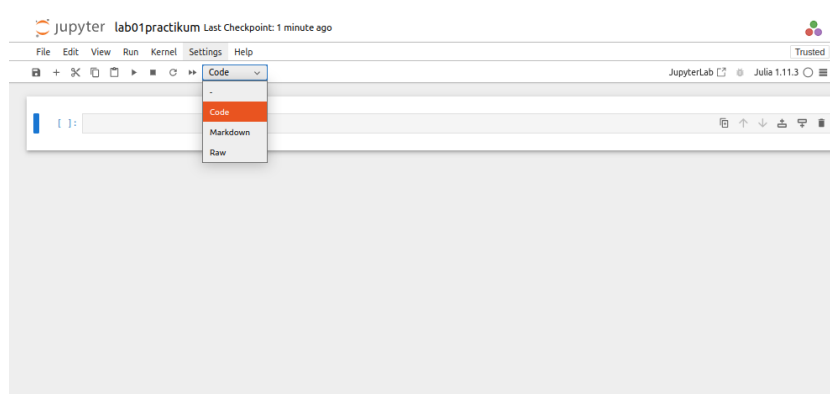


Рис. 3.6: Режимы вставки ячейки

3.2 Основы работы в блокноте Jupyter

Каждый блокнот (или консоль, или терминал, или текстовый редактор) располагается в своей вкладке в основной рабочей области. Для создания нового блокнота выбирается в меню File (рис. 3.4), далее указывается, что именно мы хотим создать. Для открытия существующего файла используются стандартные пункты меню и навигатор. Каждый файл-блокнот представляет собой текстовый файл в формате JSON с описанием всего содержимого блокнота. Обычно файл имеет расширение `.ipynb` или `.irun`. Основная концепция интерактивных блокнотов — это ячейка, содержащая отдельный фрагмент текста (или кода). Для написания текста в ячейке нужно в панели инструментов указать Markdown, для написания

элемента кода – Code (рис. 3.6). Для изменения режимов вставки ячеек можно использовать также комбинации клавиш. Для этого нужно на активной ячейке нажать **ESC**, что выведет ячейку из режима редактирования и переведёт её в командный режим, в котором есть специальные сочетания клавиш для вставки/вырезания/изменения ячеек:

- **a** или **b** – создать новую ячейку соответственно выше или ниже текущей;
- **x** – удалить ячейку;
- **z** – отмена удаления ячейки;
- **m** – перевести ячейку в режим текста;
- **y** – перевести ячейку в режим набора кода.

Для выполнения кода внутри ячейки выберите эту ячейку и нажимается **Shift + Enter** или кнопку со значком **Run** на панели инструментов. Если ячейка содержит несколько строк кода, то при выполнении этой ячейки отобразится только результат последней строки (операции). Вывод результата можно подавить, завершив строку знаком «точка с запятой».

Выполним примеры кода простейшей операцией сложения в блокноте Jupyter (рис. 3.7):

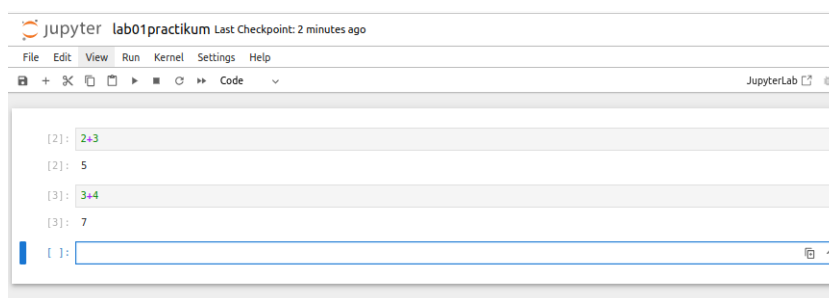


Рис. 3.7: Простейшие операции на языке Julia в Jupyter Lab

Если необходимо получить информацию по работе с какой-то незнакомой для вас функцией Julia, то можно поставить в ячейке перед названием этой функции знак вопроса (рис. 3.8):

```
[4]: ?println
search: println print sprint pointer printstyled

println(io::IO, xs...)
Print (using print) xs to io followed by a newline. If io is not supplied, prints to the default output stream 'stdout'.
See also printstyled to add colors etc.

Examples

julia> println("Hello, world")
Hello, world

julia> io = IOBuffer();
julia> println(io, "Hello", ' ', " world.")
julia> String(take!(io))
"Hello, world.\n"
```

Рис. 3.8: Пример получения информации по функции println на языке Julia в Jupyter Lab

Если требуется использовать команды из командной оболочки операционной системы, то перед соответствующей командой нужно поставить знак «точка с запятой». Например, для пользователей ОС Linux можно вывести текущую дату и имя пользователя, используя последовательно команды date и whoami. Для пользователей других ОС следует использовать команды оболочки соответствующей операционной системы (рис. 3.9):

```
[5]: ;date
Пт 12 сен 2025 21:17:28 MSK

[6]: ;whoami
lesya

[ ]:
```

Рис. 3.9: Пример получения информации о дате и пользователе ОС Linux в Jupyter Lab

Для очистки результатов выполнения ячеек следует использовать меню **Edit**(рис. 3.10):

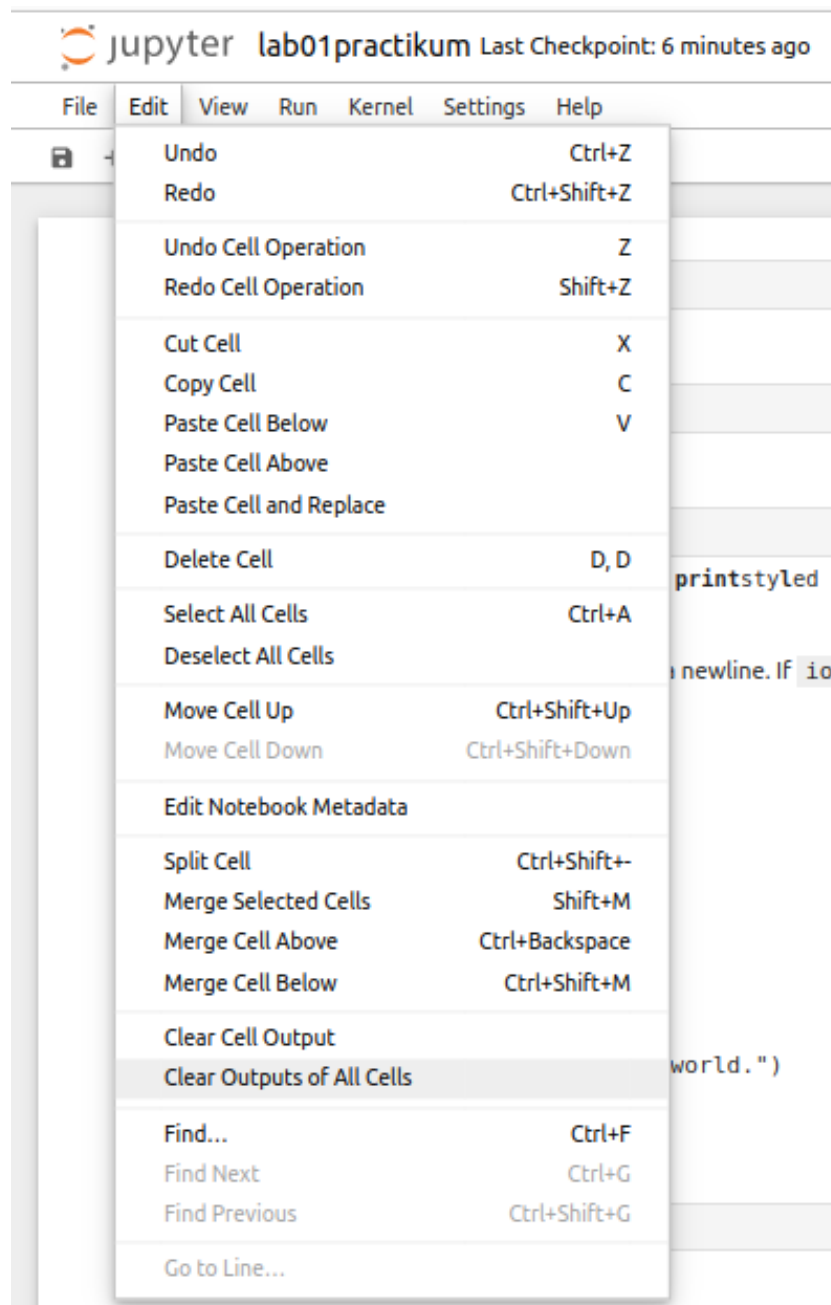
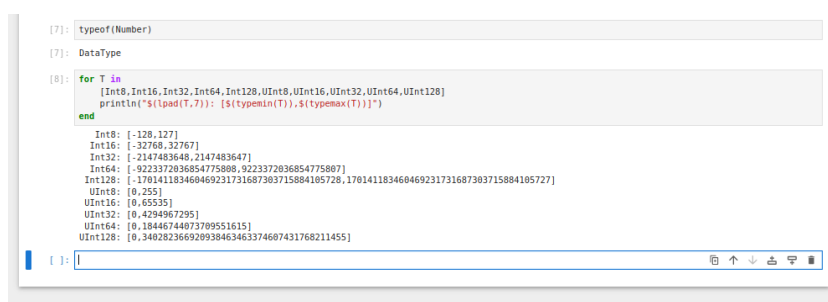


Рис. 3.10: Очистка результатов выполнения ячеек

3.3 Основы синтаксиса Julia на примерах

Далее приведены простейшие примеры с использованием синтаксиса Julia, выполненные в блокноте Jupyter Lab. Определение типа числовой величины

можно реализовать с помощью команды **typeof(Number)** (рис. 3.11). Здесь Number – конкретное число, например, 3 или 3.5, или числовой результат какой-либо операции, например, $3/3.5$, $\sqrt{3+4i}$, значение числа π . В Julia введены специальные значения **Inf**, **-Inf**, **NaN**, обозначающие бесконечность и отсутствие какого-либо значения. Такие значения могут получаться в результате операций типа деления на ноль, а также могут быть допустимой частью выражений, поскольку в языке имеют тип вещественного числа. Для определения крайних значений диапазонов целочисленных числовых величин можно воспользоваться следующим кодом (рис. 3.11):



```
[7]: typeof(Number)
[7]: DataType

[8]: for T in
      [Int8, Int16, Int32, Int64, Int128, UInt8, UInt16, UInt32, UInt64, UInt128]
      println("$(lpad(T,7)) - $(typemin(T))-$$(typemax(T))")
    end
Int8: [-128,127]
Int16: [-32768,32767]
Int32: [-2147483648,2147483647]
Int64: [-9223372036854775808,9223372036854775807]
Int128: [-170141183460469231731687303715884105728,170141183460469231731687303715884105727]
UInt8: [0,255]
UInt16: [0,65535]
UInt32: [0,4294967295]
UInt64: [0,18446744073709551615]
UInt128: [0,340282366920938463463374607431768211455]
```

Рис. 3.11: Определение крайних значений диапазонов целочисленных числовых величин и определение типа числовой величины

В результате получим минимальные и максимальные значения целочисленных типов.

В Julia преобразование типов можно реализовать или прямым указанием, например вещественное число 2.0 преобразовать в целое, а число 2 в символ: **Int64(2.0)**, **Char(2)** или использовать обобщённый оператор преобразования типов **convert()**, например: **convert(Int64, 2.0)**, **convert(Char,2)** Преобразование 1 в булевое **true**, 0 – в булевое **false**: **Bool(1)**, **Bool(0)**. Для приведения нескольких аргументов к одному типу, если это возможно, используется оператор **promote()**, например: **promote(Int8(1), Float16(4.5), Float32(4.1))** В данном выражении все аргументы оператора **promote()** в результате будут иметь тип **Float32**, в чём можно убедиться, воспользовавшись функцией определения типа **typeof** (рис. 3.12):

```
[9]: Int64(2.0), Char(2)
[9]: (2, '\x02')

[10]: convert(Int64, 2.0), convert(Char, 2)
[10]: (2, '\x02')

[11]: Bool(1), Bool(0)
[11]: (true, false)

[12]: promote(Int8(1), Float16(4.5), Float32(4.1))
[12]: (1.0f0, 4.5f0, 4.1f0)
```

Рис. 3.12: Примеры приведения аргументов к одному типу

Базовый синтаксис определения функции:

```
function <Имя> (<СписокПараметров>)
    <Действия>
end
```

Например, определим функцию $f(x)$ возведения переменной x в квадрат и возведём в квадрат число 4 (рис. 3.13):

```
function f(x)
    x^2
end
f(4)
```

Другой способ определения несложных функций (рис. 3.14):

```
<Имя> (<СписокПараметров>) = <Выражение>
```

Пример определения одномерных массивов (вектор-строка и вектор-столбец) и обращение к их вторым элементам (рис. 3.13):

```
a = [4 7 6] # вектор-строка
b = [1, 2, 3] # вектор-столбец
a[2], b[2] # вторые элементы векторов a и b
```

```

[13]: function f(x)
      x^2
    end
      f(4)

[13]: 16

[14]: a = [4 7 6] # вектор-строка
      b = [1, 2, 3] # вектор-столбец
      a[2], b[2] # вторые элементы векторов a и b

[14]: (7, 2)

[ ]: |

```

Рис. 3.13: Пример определения одномерных массивов и пример определения функций

```

[15]: g(x) = x^2

[15]: g (generic function with 1 method)

[16]: g(8)

[16]: 64

```

Рис. 3.14: Пример определения функций

Пример определения двумерного массива (матрицы) и обращение к его элементам (рис. 3.15):

```

a = 1; b = 2; c = 3; d = 4 # присвоение значений
Am = [a b; c d] # матрица 2 x 2
Am[1,1], Am[1,2], Am[2,1], Am[2,2] # элементы матрицы

```

Пример выполнения операций над массивами (aa' – транспонирование вектора) (рис. 3.15):

```

aa = [1 2]
AA = [1 2; 3 4]
aa*AA*aa

```

```
[18]: a = 1; b = 2; c = 3; d = 4 # присвоение значений
      Am = [a b; c d] # матрица 2 x 2

[18]: 2x2 Matrix{Int64}:
      1 2
      3 4

[19]: Am[1,1], Am[1,2], Am[2,1], Am[2,2] # элементы матрицы

[19]: (1, 2, 3, 4)

[25]: aa = [1 2]
      AA = [1 2; 3 4]
      aa*AA*aa'

[25]: 1x1 Matrix{Int64}:
      27

[24]: aa, AA, aa'|

[24]: ([1 2], [1 2; 3 4], [1; 2;:])

[ ]:
```

Рис. 3.15: Примеры работы с массивами

3.4 Задания для самостоятельной работы

1. Изучите документацию по основным функциям Julia для чтения / записи / вывода информации на экран: **read()**, **readline()**, **readlines()**, **readdlm()**, **print()**, **println()**, **show()**, **write()**. Приведите свои примеры их использования, поясняя особенности их применения.

Примеры

[illegible]

Рис. 3.16: Функции `read()`, `readline()`, `readlines()`

- **read()** читает весь файл, как массив байтов;
- **readline()** читает только одну строку;
- **readlines()** читает все строки.

```

[3]: using DelimitedFiles
data = readlm("Зарп/жк/сш/сatenoids_faces.csv", ',')

[3]: 362x4 Matrix{Any}:
"Vertex 1"  "Vertex 2"  "Vertex 3"  "Vertex 4"
0           1           21          20
1           2           22          21
2           3           23          22
3           4           24          23
4           5           25          24
5           6           26          25
6           7           27          26
7           8           28          27
8           9           29          28
9           10          30          29
10          11          31          30
11          12          32          31
:
367         368         388         387
368         369         389         388
369         370         390         389
370         371         391         390
371         372         392         391
372         373         393         392
373         374         394         393
374         375         395         394
375         376         396         395
376         377         397         396
377         378         398         397
378         379         399         398

[4]: print("Hello")
Hello

[5]: println("World")
World

[6]: show(stdout, 3.14159)
3.14159

[ ]: write("output.bin", data)

```

Рис. 3.17: Функции readlm(), print(), println(), show(), write()

- **readlm()** читает с разделителем;
- **print()** вывод текста по частям;
- **println()** вывод в отдельных строках.
- **show()** выводит текстовые данные;
- **write()** выводит двоичные данные.

2. Изучите документацию по функции **parse()**. Приведите свои примеры её использования, поясняя особенности её применения.

```

*[10]: x = parse{Int, "42"}# 42 (целое число)

[10]: 42

*[11]: y = parse{Float64, "3.14"}# 3.14 (число с плавающей точкой)

[11]: 3.14

*[12]: z = parse{Bool, "true"}# true (логическое значение)

[12]: true

```

Рис. 3.18: Функция parse()

- **parse()** используется для преобразования строки в число определенного типа

3. Изучите синтаксис Julia для базовых математических операций с разным типом переменных: сложение, вычитание, умножение, деление, возведение в степень, извлечение корня, сравнение, логические операции. Приведите свои примеры с пояснениями по особенностям их применения.

```
[14]: a = 2
[14]: 2

[16]: b = 4
[16]: 4

•[18]: a + b
      # сложение

[18]: 6

•[19]: a - b
      # вычитание

[19]: -2

•[20]: a * b
      # умножение

[20]: 8

•[21]: a / b
      # деление (вещественное)

[21]: 0.5

•[22]: a ÷ b
      # целочисленное деление (div(a, b))

[22]: 0

•[23]: a % b
      # остаток от деления (mod(a, b))

[23]: 2

•[24]: a ^ b
      # возведение в степень

[24]: 16

•[25]: sqrt(a)
      # квадратный корень

[25]: 1.4142135623730951
```

Рис. 3.19: Математические операции с разным типом переменных

```

•[26]: a == b
      # равно
[26]: false

•[27]: a != b
      # не равно
[27]: true

•[28]: a < b
      # меньше
[28]: true

•[29]: a > b
      # больше
[29]: false

•[30]: a <= b
      # меньше или равно
[30]: true

•[31]: a >= b
      # больше или равно
[31]: false

[39]: a = true
[39]: true

[40]: b = false
[40]: false

•[41]: !a
      # отрицание
[41]: false

•[42]: a && b
      # логическое И
[42]: false

•[43]: a || b
      # логическое ИЛИ
[43]: true

```

Рис. 3.20: Математические операции с разным типом переменных(логические операции)

4. Приведите несколько своих примеров с пояснениями с операциями над матрицами и векторами: сложение, вычитание, скалярное произведение, транспонирование, умножение на скаляр.

```

• [46]: using LinearAlgebra
A = [1 2; 3 4] # матрица 2x2
B = [5 6; 7 8]
v = [1, 2, 3] # вектор-столбец
w = [4, 5, 6]

[46]: 2x2 Matrix{Int64}:
      5 12
      21 32

• [47]: A + B
      # поэлементное сложение

[47]: 2x2 Matrix{Int64}:
      6 8
      10 12

• [48]: A - B
      # поэлементное вычитание

[48]: 2x2 Matrix{Int64}:
      -4 -4
      -4 -4

• [50]: A * B
      # матричное умножение

[50]: 2x2 Matrix{Int64}:
      19 22
      43 50

• [51]: A'
      # транспонирование

[51]: 2x2 adjoint(::Matrix{Int64}) with eltype Int64:
      1 3
      2 4

• [52]: dot(v, w)
      # скалярное произведение векторов

[52]: 32

• [53]: A .* B
      # поэлементное умножение

[53]: 2x2 Matrix{Int64}:
      5 12
      21 32

```

Рис. 3.21: Операции над матрицами и векторами

4 Выводы

В процессе выполнения данной лабораторной работы подготовила рабочее пространство и инструментарий для с языком программирования Julia, на простейших примерах познакомиться с основами синтаксиса Julia.