

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

**Дисциплина: Компьютерный практикум по статистическому анализу  
данных**

**Манаева Варвара Евгеньевна, НФИбд-01-20**

# Содержание

<b>1</b>	<b>Техническое оснащение:</b>	<b>5</b>
<b>2</b>	<b>Цели и задачи работы</b>	<b>6</b>
2.1	Цель . . . . .	6
2.2	Задачи [1] . . . . .	6
<b>3</b>	<b>Выполнение лабораторной работы</b>	<b>7</b>
3.1	Повторение задания . . . . .	7
3.2	Выполнение самостоятельной части . . . . .	8
3.2.1	Выдержки из документации [2] . . . . .	8
3.2.2	Прикладные применения . . . . .	12
<b>4</b>	<b>Выводы по проделанной работе</b>	<b>13</b>
4.1	Вывод . . . . .	13
	<b>Список литературы</b>	<b>14</b>

## Список иллюстраций

3.1	Повторение (1)	7
3.2	Повторение (2)	8
3.3	read()	8
3.4	readline()	9
3.5	readlines()	9
3.6	readrlm()	9
3.7	println()	10
3.8	print()	10
3.9	show()	10
3.10	write()	11
3.11	parse()	11
3.12	Решения (1)	12
3.13	Решения (2)	12

## Список таблиц

# 1 Техническое оснащение:

- Персональный компьютер с операционной системой Windows 10;
- Планшет для записи видеосопровождения и голосовых комментариев;
- Microsoft Teams, использующийся для записи скринкаста лабораторной работы;
- Приложение Rucharm для редактирования файлов формата *md*;
- *pandoc* для конвертации файлов отчётов и презентаций.

## 2 Цели и задачи работы

### 2.1 Цель

Подготовить рабочее пространство и инструментарий для работы с языком программирования Julia, на простейших примерах познакомиться с основами синтаксиса Julia.

### 2.2 Задачи [1]

1. Установите под свою операционную систему Julia, Jupyter (разделы 1.3.1 и 1.3.2).
2. Используя Jupyter Lab, повторите примеры из раздела 1.3.3.
3. Выполните задания для самостоятельной работы (раздел 1.3.4).

## 3 Выполнение лабораторной работы

### 3.1 Повторение задания

```
[1]: typeof(3), typeof(3.5), typeof(3/3.55), typeof(sqrt(3+4im)), typeof(pi)
[1]: (Int64, Float64, Float64, ComplexF64, Irrational{::N})

[2]: 1.0/0.0, 1.0/(-0.0), 0.0/0.0
[2]: (Inf, -Inf, NaN)

[3]: for T in [Int8,Int16,Int32,Int64,Int128,UInt8,UInt16,UInt32,UInt64,UInt128]
      println("${lpad(T,7)}: [{$(typeof(T)),$(typeof(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]

[4]: Int64(2.0), Char(2), typeof(Char(2))
[4]: (2, '\x02', Char)

[5]: convert{Int64, 2.0}, convert{Char, 2}
[5]: (2, '\x02')

[6]: typeof(promote{Int8(1), Float16(4.5), Float32(4.1)})
[6]: Tuple{Float32, Float32, Float32}

[7]: function f(x)
      x^2
    end
[7]: f (generic function with 1 method)

[8]: f(4)
[8]: 16

[9]: g(x) = x^2
```

Рис. 3.1: Повторение (1)

```

[9]: g(x) = x^2
[9]: g (generic function with 1 method)
[10]: g(8)
[10]: 64
[11]: a = [4 7 6]
      b = [1, 2, 3]
      a[2], b[2]
[11]: (7, 2)
[12]: a = 1; b = 2; c = 3; d = 4
      Am = [a b; c d]
[12]: 2x2 Matrix{Int64}:
      1  2
      3  4
[13]: Am[1,1], Am[1,2], Am[2,1], Am[2,2]
[13]: (1, 2, 3, 4)
[14]: aa = [1 2]
      AA = [1 2; 3 4]
      aa^AA^aa
[14]: 1x1 Matrix{Int64}:
      27
[15]: aa, AA, aa'
[15]: ([1 2], [1 2; 3 4], [1; 2; 1])

```

Рис. 3.2: Повторение (2)

## 3.2 Выполнение самостоятельной части

### 3.2.1 Выдержки из документации [2]

Base.read — Function

```
read(io::IO, T)
```

Read a single value of type `T` from `io`, in canonical binary representation.

Note that Julia does not convert the endianness for you. Use `ntoh` or `ltoh` for this purpose.

```
read(io::IO, String)
```

Read the entirety of `io`, as a `String` (see also `readchomp`).

Рис. 3.3: read()



#### Base.readline — Function

```
readline(io::IO=stdin; keep::Bool=false)
readline(filename::AbstractString; keep::Bool=false)
```

Read a single line of text from the given I/O stream or file (defaults to `stdin`). When reading from a file, the text is assumed to be encoded in UTF-8. Lines in the input end with `'\n'` or `"\r\n"` or the end of an input stream. When `keep` is false (as it is by default), these trailing newline characters are removed from the line before it is returned. When `keep` is true, they are returned as part of the line.

Рис. 3.4: readline()

#### Base.readlines — Function

```
readlines(io::IO=stdin; keep::Bool=false)
readlines(filename::AbstractString; keep::Bool=false)
```

Read all lines of an I/O stream or a file as a vector of strings. Behavior is equivalent to saving the result of reading `readline` repeatedly with the same arguments and saving the resulting lines as a vector of strings. See also `eachline` to iterate over the lines without reading them all at once.

Рис. 3.5: readlines()

#### DelimitedFiles.readlm — Method

```
readlm(source, delim::AbstractChar, T::Type, eol::AbstractChar; header=false, skipstart=0, ...)
```

Read a matrix from the source where each line (separated by `eol`) gives one row, with elements separated by the given delimiter. The source can be a text file, stream or byte array. Memory mapped files can be used by passing the byte array representation of the mapped segment as source.

If `T` is a numeric type, the result is an array of that type, with any non-numeric elements as `NaN` for floating-point types, or zero. Other useful values of `T` include `String`, `AbstractString`, and `Any`.

If `header` is true, the first row of data will be read as header and the tuple `(data_cells, header_cells)` is returned instead of only `data_cells`.

Specifying `skipstart` will ignore the corresponding number of initial lines from the input.

If `skipblanks` is true, blank lines in the input will be ignored.

If `use_mmap` is true, the file specified by `source` is memory mapped for potential speedups if the file is large. Default is false. On a Windows filesystem, `use_mmap` should not be set to true unless the file is only read once and is also not written to. Some edge cases exist where an OS is Unix-like but the filesystem is Windows-like.

If `quotes` is true, columns enclosed within double-quote (") characters are allowed to contain new lines and column delimiters. Double-quote characters within a quoted field must be escaped with another double-quote. Specifying `dims` as a tuple of the expected rows and columns (including header, if any) may speed up reading of large files. If `comments` is true, lines beginning with `comment_char` and text following `comment_char` in any line are ignored.

Рис. 3.6: readlm()

#### Base.println — Function

```
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.

Рис. 3.7: println()

#### Base.print — Function

```
print([io::IO], xs...)
```

Write to `io` (or to the default output stream `stdout` if `io` is not given) a canonical (un-decorated) text representation. The representation used by `print` includes minimal formatting and tries to avoid Julia-specific details.

`print` falls back to calling `show`, so most types should just define `show`. Define `print` if your type has a separate "plain" representation. For example, `show` displays strings with quotes, and `print` displays strings without quotes.

See also `println`, `string`, `printstyled`.

Рис. 3.8: print()

#### Base.show — Method

```
show(io::IO, mime, x)
```

The `display` functions ultimately call `show` in order to write an object `x` as a given `mime` type to a given I/O stream `io` (usually a memory buffer), if possible. In order to provide a rich multimedia representation of a user-defined type `T`, it is only necessary to define a new `show` method for `T`, via: `show(io, ::MIME"mime", x::T) = ...`, where `mime` is a MIME-type string and the function body calls `write` (or similar) to write that representation of `x` to `io`. (Note that the `MIME""` notation only supports literal strings; to construct MIME types in a more flexible manner use `MIME{Symbol{""}}`.)

For example, if you define a `MyImage` type and know how to write it to a PNG file, you could define a function `show(io, ::MIME"image/png", x::MyImage) = ...` to allow your images to be displayed on any PNG-capable `AbstractDisplay` (such as `IJulia`). As usual, be sure to `import Base.show` in order to add new methods to the built-in Julia function `show`.

Technically, the `MIME"mime"` macro defines a singleton type for the given `mime` string, which allows us to exploit Julia's dispatch mechanisms in determining how to display objects of any given type.

The default MIME type is `MIME"text/plain"`. There is a fallback definition for `text/plain` output that calls `show` with 2 arguments, so it is not always necessary to add a method for that case. If a type benefits from custom human-readable output though, `show(io, ::MIME"text/plain", x::T)` should be defined. For example, the `Day` type uses `1 day` as the output for the `text/plain` MIME type, and `Day(1)` as the output of 2-argument `show`.

Рис. 3.9: show()

Base.write — Function

```
write(io::IO, x)
write(filename::AbstractString, x)
```

Write the canonical binary representation of a value to the given I/O stream or file. Return the number of bytes written into the stream. See also `print` to write a text representation (with an encoding that may depend upon `io`).

The endianness of the written value depends on the endianness of the host system. Convert to/from a fixed endianness when writing/reading (e.g. using `htol` and `ltoh`) to get results that are consistent across platforms.

You can write multiple values with the same `write` call. i.e. the following are equivalent:

```
write(io, x, y...)
write(io, x) + write(io, y...)
```

Рис. 3.10: write()

Base.parse — Function

```
parse(type, str; base)
```

Parse a string as a number. For `Integer` types, a base can be specified (the default is 10). For floating-point types, the string is parsed as a decimal floating-point number. `Complex` types are parsed from decimal strings of the form `"R±Iim"` as a `Complex(R,I)` of the requested type; `"i"` or `"j"` can also be used instead of `"im"`, and `"R"` or `"Iim"` are also permitted. If the string does not contain a valid number, an error is raised.

**Julia 1.1**

`parse{Bool, str}` requires at least Julia 1.1.

Рис. 3.11: parse()

## 3.2.2 Прикладные применения

```
[16]: write("my_file.txt", "Удивительное рядом!\nДостаточно просто протянуть руку!")
      read("my_file.txt", String)

[16]: "Удивительное рядом!\nДостаточно просто протянуть руку!"

[17]: readline("my_file.txt")

[17]: "Удивительное рядом!"

[18]: readlines("my_file.txt")

[18]: 2-element Vector{String}:
      "Удивительное рядом!"
      "Достаточно просто протянуть руку!"

[19]: print("Julia is a programming language")
      print("Julia is a programming language")
      Julia is a programming languageJulia is a programming language

[20]: println("Julia is a programming language")
      println("Julia is a programming language")
      Julia is a programming language
      Julia is a programming language

[20]: struct November
      n::Int
      end

Base.show{IO{IO, ::MIME"text/plain"}, d::November} = print(io, d.n, " ноября")
November(11)

[20]: 11 ноября

[22]: open("delim_file.txt", "w") do f
      write(f, "1,2\n3,4\n5,6\n7,8")
      end

[22]: 15
```

Рис. 3.12: Решения (1)

```
[23]: using DelimitedFiles
      readlm("delim_file.txt", ',', Float64)

[23]: 4x2 Matrix{Float64}:
      1.0  2.0
      3.0  4.0
      5.0  6.0
      7.0  8.0

[24]: parse{Int, "afc", base = 16}

[24]: 2812

[25]: 4*5, [1 2] * [2 3], [1, 2].*3, [1 2] * [1, 2], [10 5]./5

[25]: (9, [3 5], [3, 6], [5], [2.0 1.0])

[26]: [1 3]', [2, 4]', [1 2; 3 4]'

[26]: ([1; 3;], [2 4], [1 2; 3 4])
```

Рис. 3.13: Решения (2)

## 4 Выводы по проделанной работе

### 4.1 Вывод

В результате выполнения работы мы на простейших примерах ознакомились с основами синтаксиса языка Julia.

Были записаны скринкасты выполнения и защиты лабораторной работы.

Ссылки на скринкасты:

- Выполнение, Youtube
- Выполнение, Rutube
- Защита презентации, Youtube
- Защита презентации, Rutube

## Список литературы

1. Задание по выполнению лабораторной работы № 1 [Электронный ресурс]. Российский Университет Дружбы Народов имени Патрису Лумумбы, 2023. URL: <https://esystem.rudn.ru/mod/resource/view.php?id=1069827>.
2. Julia official documentation [Электронный ресурс]. 2023. URL: <https://docs.julialang.org/en/v1/>.