

Лабораторная работа №1

Julia. Установка и настройка. Основные принципы.

Герра Гарсия Паола Валентина

Российский университет дружбы народов, Москва, Россия

Информация

- Герра Гарсия Паола Валентина
- студентка
- Российский университет дружбы народов
- 1032225472@pfur.ru

Цель работы

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

Задание

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

Выполнение лабораторной работы

Выполнение лабораторной работы

Рис. 1: Запуск Julia

Простейшие примеры на Julia

Рис. 1.4. Пример получения информации о дате и выполнение OS Linux в Jupyter

С основами синтаксиса языка Julia можно ознакомиться в источниках [1–5]. Для примеров приведены примеры с использованием синтаксиса языка и более подробно в [6].

Определение типа чистовой величины:

```
typeof(Numbers)
```

Здесь `Numbers` – это строка, например, `3` или `3.5`, или чистой результат любой операции, например, `sqrt(5)`, `sqrt(3) + 4i`, комплексные числа и т.д.

В языке имеются специальные значения `Inf`, `-Inf`, `Nan`, обозначающие бесконечность и неопределенность соответственно. Важно помнить, что в языке есть тип `NaN` (не-число), а также нужно быть достаточно осторожным с числом `Inf`, поскольку он может привести к ошибкам (см. рис. 1.5).

Для выполнения приведенных примеров необходимо использовать языковую среду Julia, которая доступна для загрузки из официального сайта [JuliaLang.org](https://julialang.org).

```
for T in
    [Int8, Int16, Int32, Int64, UInt8, UInt16, UInt32, UInt64, UInt128]
        println("Type $T")
    end
```

В результате получим информативные и максимальные значения целочисленных типов (см. рис. 1.5).

В языке преобразование типов можно реализовать при помощи указателей, например, вещественное число 1.0 преобразовать в целое, а число 2 в символ:

```
out[13]: 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"
```

```
In [14]: println("Hello, world")
Hello, world
```

```
In [15]: io = IOBuffer()
Out[15]: IOBuffer(data=UInt8[], readable=true, writable=true, seekable=true,
append=false, size=0, maxsize=Inf, ptr=1, mark=-1)
```

```
In [28]: println(io, "Hello, world")
String(take!(io))
Out[28]: "Hello, world\n"
```

Рис. 2: Выполнение примеров из лабораторной

Простейшие примеры на Julia

The image shows two side-by-side screenshots of a Julia notebook interface. Both windows have a title bar with tabs for 'Method' and 'Op', and a toolbar with icons for file operations, run, and code.

Left Notebook (Screenshot from lab1_intro_julia.pdf):

- In [1]:** `convert(Int64, 2.0), convert(Char, 2)`
Output: `Int128: (-170141183468469231731687383715884105728, 170141183468469231731687383715884105727) UInt32: [0,65535] UInt32: [0,4294967295] UInt64: [0,18446744073798551615] UInt128: [0,3402823660293846346374687431768211455]
- In [2]:** `Int(2.0), Char(2), typeof(Char(2))`
Output: `(2, '\x02', Char)
- In [3]:** `convert(Int64, 2.0), convert(Char, 2)`
Output: `(2, '\x02')`
- In [4]:** `typeof(promote(Int8(1), Float16(4.5), Float32(4.1)))`
Output: `Tuple{Float32, Float32, Float32}`
- In [5]:** `function f(x)
 x^2
end`
Output: `f (generic function with 1 method)`
- In [6]:** `f(4)`
Output: `16`
- In [7]:** `g(x)=x^2`
Output: `g (generic function with 1 method)`
- In [8]:** `g(8)`
Output: `64`

Right Notebook (Лаб1_статистический_анализ):

- In [1]:** `convert(Int64, 2.0), convert(Char, 2)`
Output: `Int128: (-170141183468469231731687383715884105728, 170141183468469231731687383715884105727) UInt32: [0,65535] UInt32: [0,4294967295] UInt64: [0,18446744073798551615] UInt128: [0,3402823660293846346374687431768211455]
- In [2]:** `Int(2.0), Char(2), typeof(Char(2))`
Output: `(2, '\x02', Char)
- In [3]:** `convert(Int64, 2.0), convert(Char, 2)`
Output: `(2, '\x02')`
- In [4]:** `typeof(promote(Int8(1), Float16(4.5), Float32(4.1)))`
Output: `Tuple{Float32, Float32, Float32}`
- In [5]:** `function f(x)
 x^2
end`
Output: `f (generic function with 1 method)`
- In [6]:** `f(4)`
Output: `16`
- In [7]:** `g(x)=x^2`
Output: `g (generic function with 1 method)`
- In [8]:** `g(8)`
Output: `64`

Рис. 3: Выполнение примеров из лабораторной

Простейшие примеры на Julia

The image shows two side-by-side Julia environments. The left window is a browser-based interface titled 'jupyter' with the URL 'localhost:8888/notebooks/lab1_statis...'. It displays a Jupyter notebook cell with the following code:

```
a = [1; 2; 3; 4]; b = [1; 2; 3; 4] # a -> прямые элементы  
Am = [a; b]; AA = Am' # матрица J, x = AA  
Am[1,1], Am[1,2], Am[2,1], Am[2,2] # элементы матрицы  
# прямое выполнение операций над массивами (aa' -> транспонирование вектора  
# прямое выполнение операций над матрицами (AA* -> транспонирование матрицы)  
aa=Am*a  
aa*AA*aa  
  
a = [1; 2; 3; 4]; b = [1; 2; 3; 4]; c = [1; 2; 3; 4]; d = 4  
a[1,1], b[1,1], c[1,1], d # прямые элементы  
M1, M2 # прямые элементы-матрицы, x = b  
  
Am[1,1], Am[1,2], Am[2,1], Am[2,2] # элементы матрицы  
(1, 1), (1, 2), (2, 1), (2, 2)  
  
m1, m2, m3  
(1, 1), (2, 1), (3, 1), (1, 2)
```

The right window is a standard terminal-based Julia environment with the title 'jupyter Lab1_статистический_анализ' and the command 'Julia 1.12.0'. It shows the following code execution:

```
In [33]: 64  
Out[33]: 64  
  
In [34]: a = [1 2 3]#vector fila  
          a[1, 2, 3]#vector columna  
          a[1], b[2]#segundo elemento del vector  
Out[34]: (1, 2)  
  
In [35]: a = [1; 2; 3; 4];  
          Am = [a; b; c; d] #matriz 2x2  
Out[35]: 2x2 Matrix{Int64}:  
          1 2  
          3 4  
  
In [36]: Am[1,1], Am[1,2], Am[2,1], Am[2,2] #elementos de la matriz am  
Out[36]: (1, 2, 3, 4)  
  
In [37]: aa = [1; 2]  
          AA = aa*aa'  
          aa*AA*aa'  
Out[37]: 1x1 Matrix{Int64}:  
          27  
  
In [38]: aa, AA, aa'  
Out[38]: ([1; 2], [1; 2; 3; 4], [1; 2; 2])  
  
In [ ]:
```

Рис. 4: Выполнение примеров из лабораторной

Задание №1

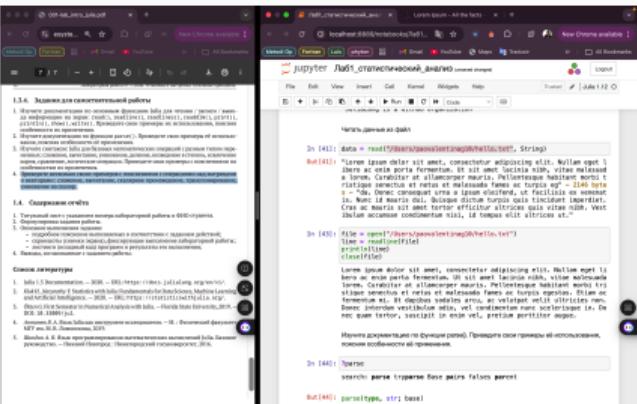


Рис. 5: Чтение файла

Задание №1

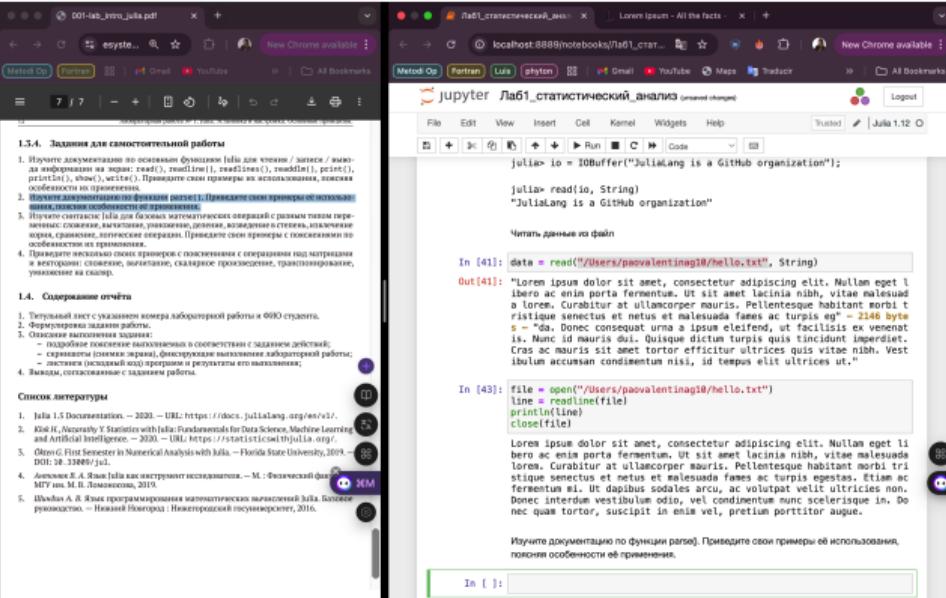


Рис. 6: Вывод на печать

Задание №1

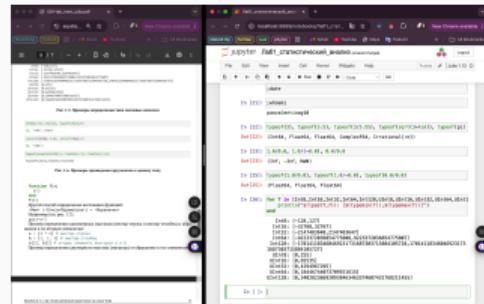


Рис. 7: Вывод на печать

Задание №1

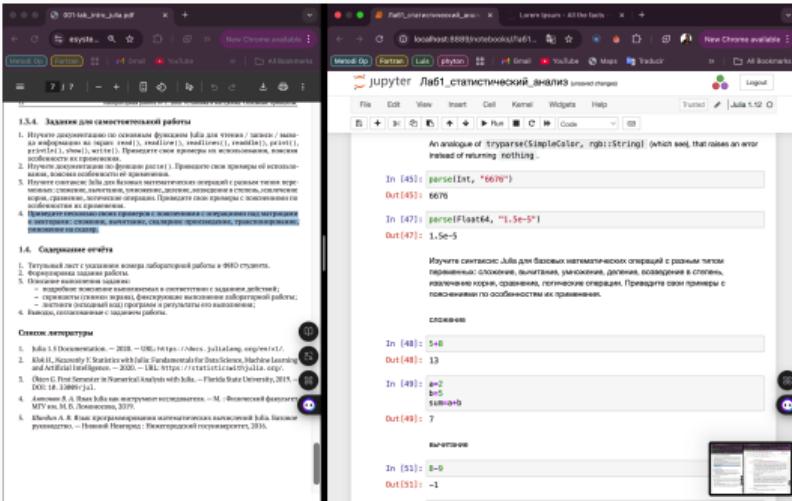


Рис. 8: Команда записи

Задание №2

```
In [44]: ?parse
search: parse tryparse Base pairs falses parent

Out[44]: \begin{verbatim}
\begin{verb}
parse(type, str; base)
\end{verb}

Parse a string as a number. For \texttt{\{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. \texttt{\{Complex\}} types are parsed from decimal strings of the form \texttt{\{"R\pm im"\}} as a \texttt{\{Complex(R,I)\}} of the requested type; \texttt{\{"i"\}} or \texttt{\{"r"\}} can also be used instead of \texttt{\{"im"\}}, and \texttt{\{"R"\}} or \texttt{\ {"im"\}} are also permitted. If the string does not contain a valid number, an error is raised.
\begin{quote}
\begin{verb}
\Textbf{compat}
\end{verb}
Julia 1.1

\texttt{\{parse(Bool, str)\}} requires at least Julia 1.1.

\end{quote}

\section{Examples}
\begin{verb}
\begin{verb}
julia> parse(int, "1234")
1234

julia> parse(int, "1234", base = 5)
194

julia> parse(int, "afc", base = 16)
2812

julia> parse(Float64, "1.2e-3")
0.0012

julia> parse(Complex{Float64}, "3.2e-1 + 4.5im")
0.32 + 4.5im
\end{verb}
\end{verb}

\rule{\textwidth}{1pt}

\begin{verb}
\begin{verb}
\end{verb}
\end{verb}
\end{verb}
```

Рис. 9: Документация по функции parse()

Задание №2

```
In [45]: parse(Int, "6676")
Out[45]: 6676

In [47]: parse(Float64, "1.5e-5")
Out[47]: 1.5e-5
```

Рис. 10: Примеры использования функции parse()

Задание №3

The screenshot shows a Jupyter Notebook interface with several code cells and their corresponding outputs:

- In [54]: $a=4$
Out[54]: -1
- In [55]: $[a,b]$
Out[55]: $\text{diff}[a,b]$
Out[55]: -1
- умножение
- In [56]: $a*b$
Out[56]: 36
- деление
- In [57]: a/b
Out[57]: 4.075
- степень
- In [58]: a^b
Out[58]: 3744881
- In [59]: $\sqrt{58}$
Out[59]: 7.615773185863989
- In [60]: a/b
Out[60]: 4.075

Рис. 11: Примеры базовых математических операций

Задание №3

Рис. 12: Примеры базовых математических операций

Задание №4

The screenshot shows a Jupyter Notebook interface with the title "jupyter Lab1_статистический_анализ Last Checkpoint: el sábado pasado a las 20:01 (outsaved)". The notebook contains several code cells demonstrating matrix operations:

- In [61]:

```
a=[1 2; 3 4]
b=[0 7; 8 9]
a+b
```

Out[61]:
2x2 Matrix{Int64}:
7 9
11 13
- In [62]:

```
a-b
```

Out[62]:
2x2 Matrix{Int64}:
-5 -5
-5 -5
- In [63]:

```
using LinearAlgebra
v1=[1,2,3]
v2=[4,5,6]
dot(v1,v2)
```

Out[63]: 32
- In [64]:

```
a'
```

Out[64]:
2x2 adjoint(::Matrix{Int64}) with eltype Int64:
1 3
2 4
- In [65]:

```
2*a
```

Out[65]:
2x2 Matrix{Int64}:
2 4
6 8

Рис. 13: примеры операций над матрицами

Задание №4

```
2> [1]: using LinuxMatrix
   [1]: vcl::v1, v2, v3
   [1]: vcl::v4, v5
2> [2]: <empty>
3> [3]: 
2> [4]: 
3> [5]: 2x 2x(2x2) = 2x adjoint((Matrix2x2)) with entries float64
   [5]: 1  1
   [5]: 1  1
2> [6]: -2x2
3> [7]: 2x 2x(2x2) = 2x matrix2x2(float64)
   [7]: 1  1
   [7]: 1  1
```

Рис. 14: примеры операций над векторами

Выводы

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

Список литературы

1. JuliaLang [Электронный ресурс]. 2024 JuliaLang.org contributors. URL: <https://julialang.org/> (дата обращения: 11.10.2024).
2. Julia 1.11 Documentation [Электронный ресурс]. 2024 JuliaLang.org contributors. URL: <https://docs.julialang.org/en/v1/> (дата обращения: 11.10.2024).