Лабораторная работа №1

Julia. Установка и настройка. Основные принципы.

Тазаева Анастасия Анатольевна

Содержание

Список иллюстраций

Список таблиц

1 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 (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 Установка необходимого программного обеспечения

1. Для дальнейшей работы с лабораторными занятиями нам понадобится установить несколько приложений. Для этого установим Chocolatey (рис. 3.1), Far (рис. 3.2), Notepad++ (рис. 3.3), Julia (рис. 3.4), Anaconda3 (рис. 3.5) с помощью комманд:

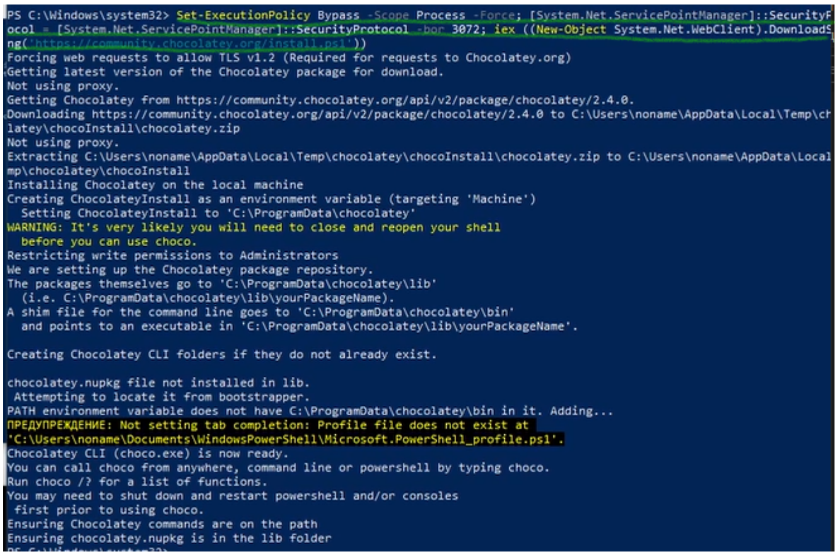
```
Set-ExecutionPolicy Bypass -Scope Process -Force; [System.Net.ServicePointManager]::Se
bor 3072; iex ((New-Object System.Net.WebClient).DownloadString('https://community.cho
```

```
choco install far
```

```
choco install notepadplusplus
```

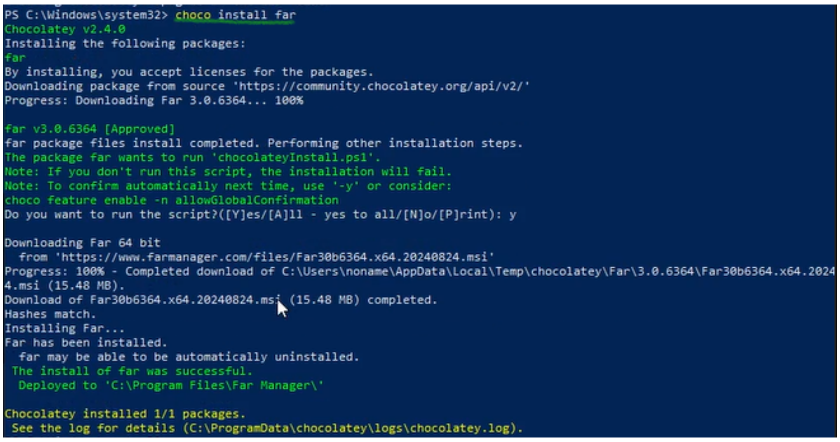
```
choco install julia
```

```
choco install anaconda3
```



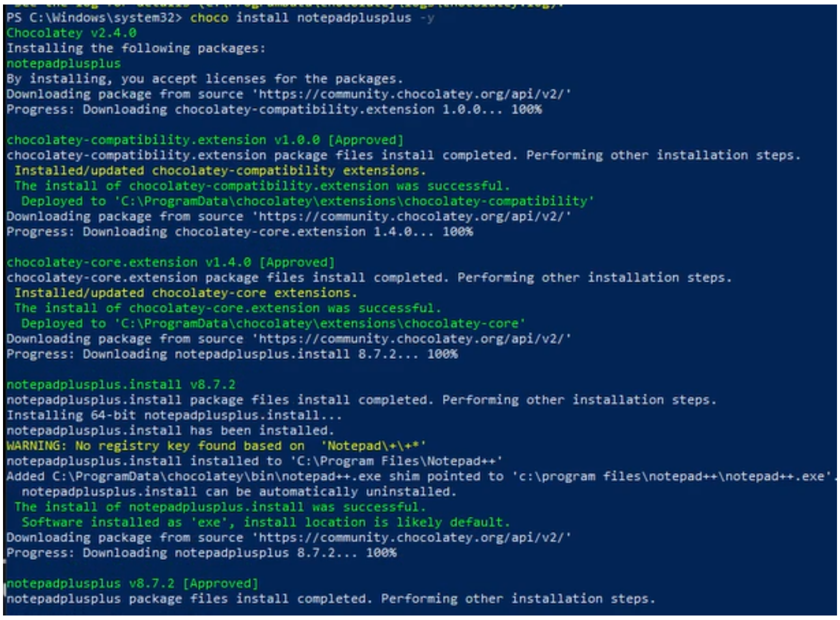
```
PS C:\Windows\system32> Set-ExecutionPolicy Bypass -Scope Process -Force; [System.Net.ServicePointManager]::SecurityP
ocol = [System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object System.Net.WebClient).DownloadS
ng('https://community.chocolatey.org/install.ps1'))
Forcing web requests to allow TLS v1.2 (Required for requests to Chocolatey.org)
Getting latest version of the Chocolatey package for download.
Not using proxy.
Getting Chocolatey from https://community.chocolatey.org/api/v2/package/chocolatey/2.4.0.
Downloading https://community.chocolatey.org/api/v2/package/chocolatey/2.4.0 to C:\Users\noname\AppData\Local\Temp\ch
latey\chocoInstall\chocolatey.zip
Not using proxy.
Extracting C:\Users\noname\AppData\Local\Temp\chocolatey\chocoInstall\chocolatey.zip to C:\Users\noname\AppData\Local
mp\chocolatey\chocoInstall
Installing Chocolatey on the local machine
Creating ChocolateyInstall as an environment variable (targeting 'Machine')
  Setting ChocolateyInstall to 'C:\ProgramData\chocolatey'
WARNING: It's very likely you will need to close and reopen your shell
  before you can use choco.
Restricting write permissions to Administrators
We are setting up the Chocolatey package repository.
The packages themselves go to 'C:\ProgramData\chocolatey\lib'
  (i.e. C:\ProgramData\chocolatey\lib\yourPackageName).
A shim file for the command line goes to 'C:\ProgramData\chocolatey\bin'
  and points to an executable in 'C:\ProgramData\chocolatey\lib\yourPackageName'.

Creating Chocolatey CLI folders if they do not already exist.

chocolatey.nupkg file not installed in lib.
 Attempting to locate it from bootstrapper.
PATH environment variable does not have C:\ProgramData\chocolatey\bin in it. Adding...
ПРЕДУПРЕЖДЕНИЕ: Not setting tab completion: Profile file does not exist at
'C:\Users\noname\Documents\WindowsPowerShell\Microsoft.PowerShell_profile.ps1'.
Chocolatey CLI (choco.exe) is now ready.
You can call choco from anywhere, command line or powershell by typing choco.
Run choco /? for a list of functions.
You may need to shut down and restart powershell and/or consoles
 first prior to using choco.
Ensuring Chocolatey commands are on the path
Ensuring chocolatey.nupkg is in the lib folder
```

Рис. 3.1: Установка Chocolatey



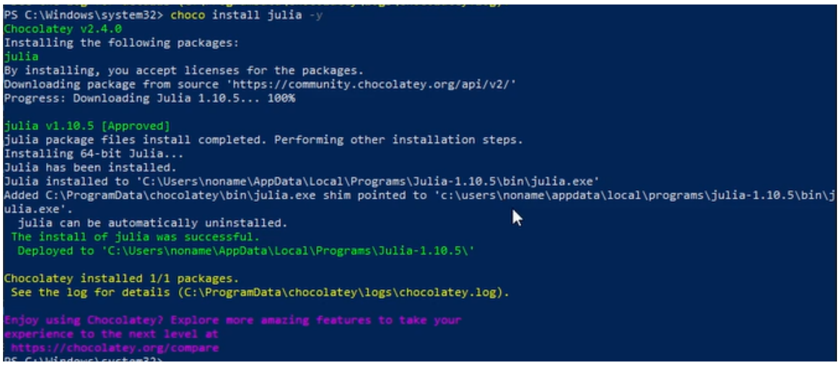
```
PS C:\Windows\system32> choco install far
Chocolatey v2.4.0
Installing the following packages:
far
By installing, you accept licenses for the packages.
Downloading package from source 'https://community.chocolatey.org/api/v2/'
Progress: Downloading Far 3.0.6364... 100%

far v3.0.6364 [Approved]
far package files install completed. Performing other installation steps.
The package far wants to run 'chocolateyInstall.ps1'.
Note: If you don't run this script, the installation will fail.
Note: To confirm automatically next time, use '-y' or consider:
choco feature enable -n allowGlobalConfirmation
Do you want to run the script?([Y]es/[A]ll - yes to all/[N]o/[P]rint): y

Downloading Far 64 bit
  from 'https://www.farmanager.com/files/Far30b6364.x64.20240824.msi'
Progress: 100% - Completed download of C:\Users\noname\AppData\Local\Temp\chocolatey\Far\3.0.6364\Far30b6364.x64.2024
4.msi (15.48 MB).
Download of Far30b6364.x64.20240824.msi (15.48 MB) completed.
Hashes match.
Installing Far...
Far has been installed.
  far may be able to be automatically uninstalled.
 The install of far was successful.
  Deployed to 'C:\Program Files\Far Manager\'

Chocolatey installed 1/1 packages.
 See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log).
```

Рис. 3.2: Установка Far



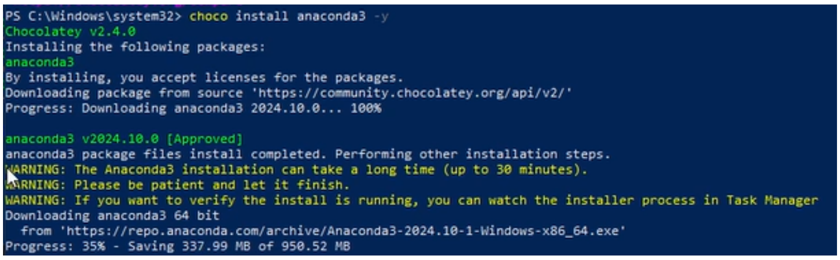
```
PS C:\Windows\system32> choco install notepadplusplus -y
Chocolatey v2.4.0
Installing the following packages:
notepadplusplus
By installing, you accept licenses for the packages.
Downloading package from source 'https://community.chocolatey.org/api/v2/'
Progress: Downloading chocolatey-compatibility.extension 1.0.0... 100%

chocolatey-compatibility.extension v1.0.0 [Approved]
chocolatey-compatibility.extension package files install completed. Performing other installation steps.
 Installed/updated chocolatey-compatibility extensions.
 The install of chocolatey-compatibility.extension was successful.
  Deployed to 'C:\ProgramData\chocolatey\extensions\chocolatey-compatibility'
Downloading package from source 'https://community.chocolatey.org/api/v2/'
Progress: Downloading chocolatey-core.extension 1.4.0... 100%

chocolatey-core.extension v1.4.0 [Approved]
chocolatey-core.extension package files install completed. Performing other installation steps.
 Installed/updated chocolatey-core extensions.
 The install of chocolatey-core.extension was successful.
  Deployed to 'C:\ProgramData\chocolatey\extensions\chocolatey-core'
Downloading package from source 'https://community.chocolatey.org/api/v2/'
Progress: Downloading notepadplusplus.install 8.7.2... 100%

notepadplusplus.install v8.7.2
notepadplusplus.install package files install completed. Performing other installation steps.
Installing 64-bit notepadplusplus.install...
notepadplusplus.install has been installed.
WARNING: No registry key found based on  'Notepad\+\+*'
notepadplusplus.install installed to 'C:\Program Files\Notepad++'
Added C:\ProgramData\chocolatey\bin\notepad++.exe shim pointed to 'c:\program files\notepad++\notepad++.exe'.
  notepadplusplus.install can be automatically uninstalled.
 The install of notepadplusplus.install was successful.
  Software installed as 'exe', install location is likely default.
Downloading package from source 'https://community.chocolatey.org/api/v2/'
Progress: Downloading notepadplusplus 8.7.2... 100%

notepadplusplus v8.7.2 [Approved]
notepadplusplus package files install completed. Performing other installation steps.
```

Рис. 3.3: Установка Notepad++



```
PS C:\Windows\system32> choco install julia -y
Chocolatey v2.4.0
Installing the following packages:
julia
By installing, you accept licenses for the packages.
Downloading package from source 'https://community.chocolatey.org/api/v2/'
Progress: Downloading Julia 1.10.5... 100%

julia v1.10.5 [Approved]
julia package files install completed. Performing other installation steps.
Installing 64-bit Julia...
Julia has been installed.
Julia installed to 'C:\Users\noname\AppData\Local\Programs\Julia-1.10.5\bin\julia.exe'
Added C:\ProgramData\chocolatey\bin\julia.exe shim pointed to 'c:\users\noname\appdata\local\programs\julia-1.10.5\bin\j
ulia.exe'.
  julia can be automatically uninstalled.
 The install of julia was successful.
  Deployed to 'C:\Users\noname\AppData\Local\Programs\Julia-1.10.5\'

Chocolatey installed 1/1 packages.
 See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log).

Enjoy using Chocolatey? Explore more amazing features to take your
experience to the next level at
 https://chocolatey.org/compare
```

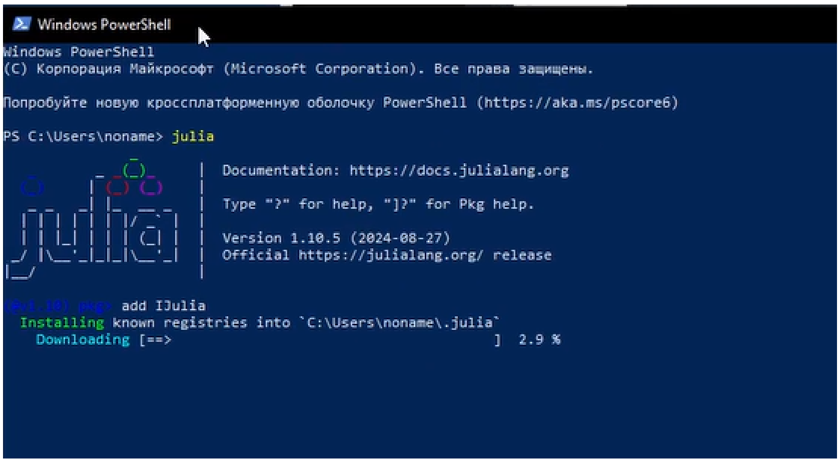
Рис. 3.4: Установка Julia



```
PS C:\Windows\system32> choco install anaconda3 -y
Chocolatey v2.4.0
Installing the following packages:
anaconda3
By installing, you accept licenses for the packages.
Downloading package from source 'https://community.chocolatey.org/api/v2/'
Progress: Downloading anaconda3 2024.10.0... 100%

anaconda3 v2024.10.0 [Approved]
anaconda3 package files install completed. Performing other installation steps.
WARNING: The Anaconda3 installation can take a long time (up to 30 minutes).
WARNING: Please be patient and let it finish.
WARNING: If you want to verify the install is running, you can watch the installer process in Task Manager
Downloading anaconda3 64 bit
  from 'https://repo.anaconda.com/archive/Anaconda3-2024.10-1-Windows-x86_64.exe'
Progress: 35% - Saving 337.99 MB of 950.52 MB
```

Рис. 3.5: Установка Anaconda3

2. Также установим пакет IJulia (рис. 3.6) для работы в Jupyter :

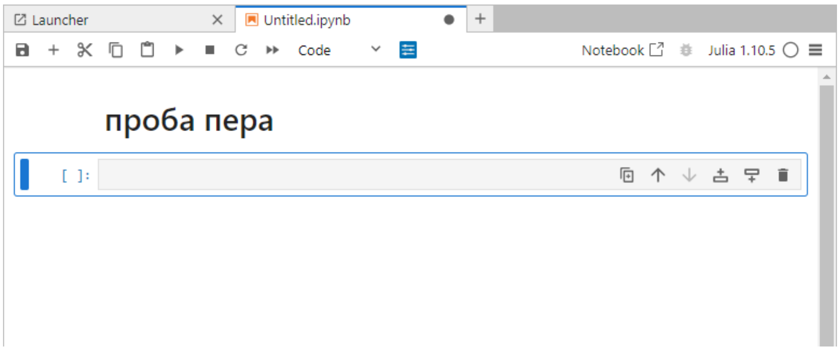


```
Windows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

Попробуйте новую кроссплатформенную оболочку PowerShell (https://aka.ms/pscore6)

PS C:\Users\noname> julia
               _
   _       _ _(_)_     |  Documentation: https://docs.julialang.org
  (_)     | (_) (_)    |
   _ _   _| |_  __ _   |  Type "?" for help, "]?" for Pkg help.
  | | | | | | |/ _` |  |
  | | |_| | | | (_| |  |  Version 1.10.5 (2024-08-27)
 _/ |\__'_|_|_|\__'_|  |  Official https://julialang.org/ release
|__/                   |

(@v1.10) pkg> add IJulia
  Installing known registries into `C:\Users\noname\.julia`
    Downloading [==>                                        ]  2.9 %
```

Рис. 3.6: Установка IJulia

3.2 Основы работы в блокноте Jupyter

1. Создала файл, указала ядро Julia 1.10.15 (рис. 3.7), далее опробовала комбинации клавиш, представленные в файле лаб.работы

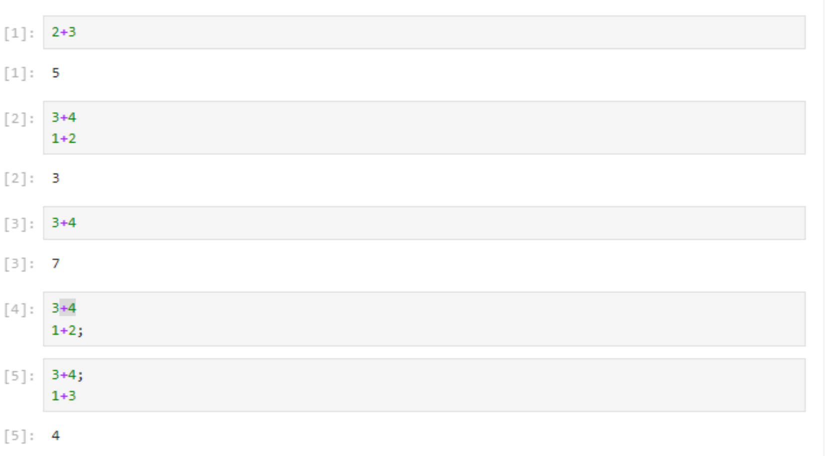


проба пера

```
[ ]:
```

Рис. 3.7: Создание файла. Разминка

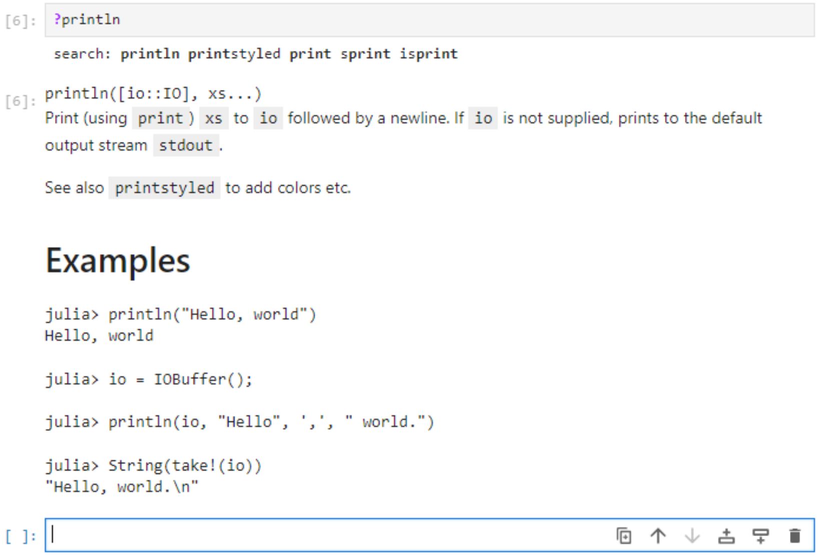
2. Выполнила простейшие операции на языке Julia в Jupyter Lab (рис. 3.8)



```
[1]: 2+3
[1]: 5
[2]: 3+4
     1+2
[2]: 3
[3]: 3+4
[3]: 7
[4]: 3+4
     1+2;
[5]: 3+4;
     1+3
[5]: 4
```

Рис. 3.8: Простейшие операции на языке Julia в Jupyter

3. С помощью команды ? получила информацию по функции (рис. 3.9):



```
[6]: ?println
     search: println printstyled print sprint isprint

[6]: 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.

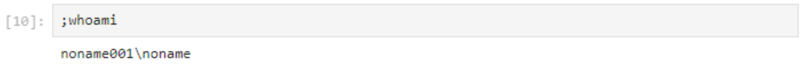
Examples

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

julia> io = IOBuffer();

julia> println(io, "Hello", ',', " world.")

julia> String(take!(io))
"Hello, world.\n"
```

```
[ ]:
```

Рис. 3.9: Пример получения информации по функции println на языке Julia в Jupyter Lab

4. С помощью команды ; получила информацию о пользователе. Эта команда может использовать команды из командной оболочки вашей операционной системы (рис. 3.10):

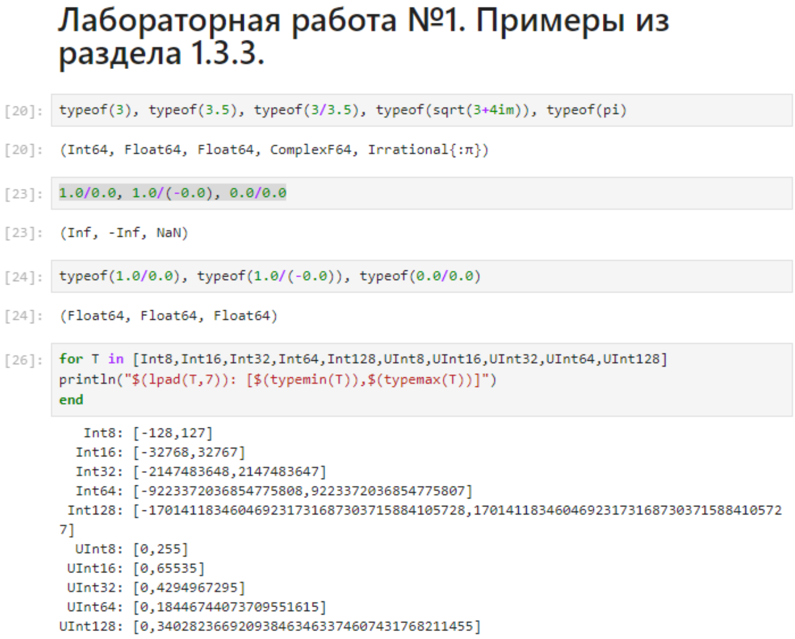


```
[10]: ;whoami

noname001\noname
```

Рис. 3.10: Пример получения информации о пользователе на языке Julia в Jupyter Lab

3.3 Повторение примеров из раздела 1.3.3

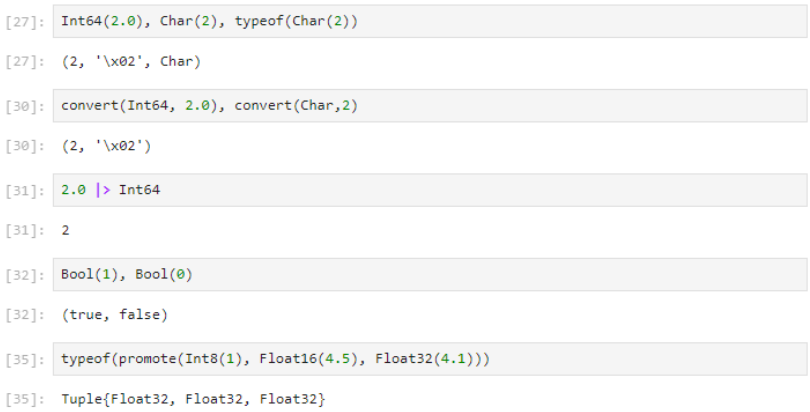
1. Узнала как определять тип данных (рис. 3.11), как конвертировать данные (рис. 3.12), как создавать функции (рис. 3.13), как создавать матрицы, также провела операции над матрицами (рис. 3.14).



Лабораторная работа №1. Примеры из раздела 1.3.3.

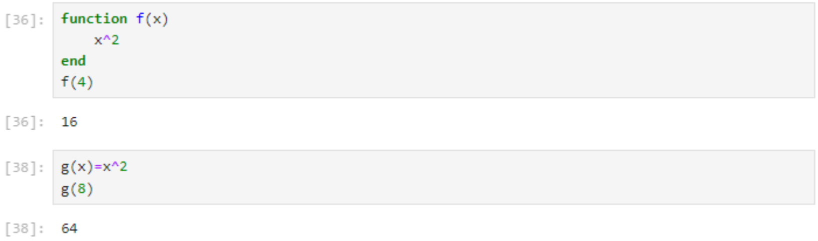
```
[20]: typeof(3), typeof(3.5), typeof(3/3.5), typeof(sqrt(3+4im)), typeof(pi)

[20]: (Int64, Float64, Float64, ComplexF64, Irrational{:π})

[23]: 1.0/0.0, 1.0/(-0.0), 0.0/0.0

[23]: (Inf, -Inf, NaN)

[24]: typeof(1.0/0.0), typeof(1.0/(-0.0)), typeof(0.0/0.0)

[24]: (Float64, Float64, Float64)

[26]: 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: Примеры определения типа числовых величин



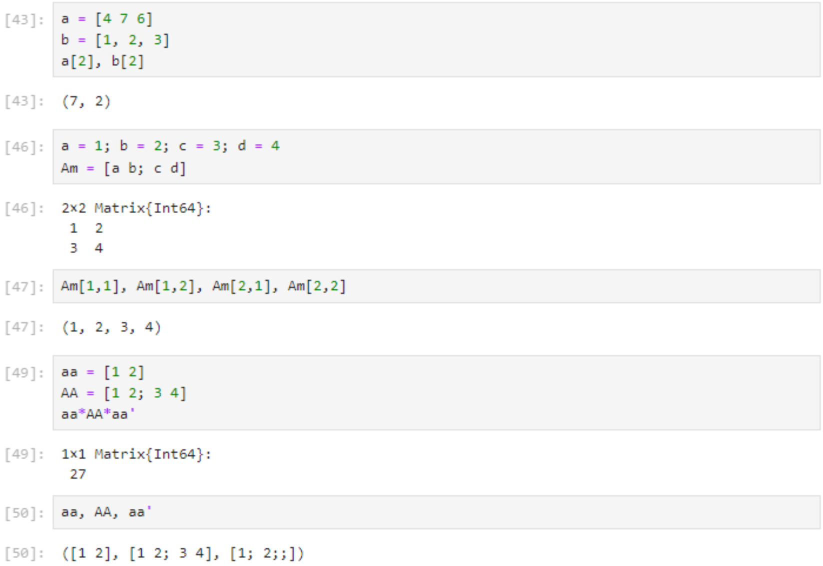
```
[27]: Int64(2.0), Char(2), typeof(Char(2))

[27]: (2, '\x02', Char)

[30]: convert(Int64, 2.0), convert(Char,2)

[30]: (2, '\x02')

[31]: 2.0 |> Int64

[31]: 2

[32]: Bool(1), Bool(0)

[32]: (true, false)

[35]: typeof(promote(Int8(1), Float16(4.5), Float32(4.1)))

[35]: Tuple{Float32, Float32, Float32}
```

Рис. 3.12: Примеры приведения аргументов к одному типу



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

[36]: 16

[38]: g(x)=x^2
      g(8)

[38]: 64
```

Рис. 3.13: Примеры определения функций



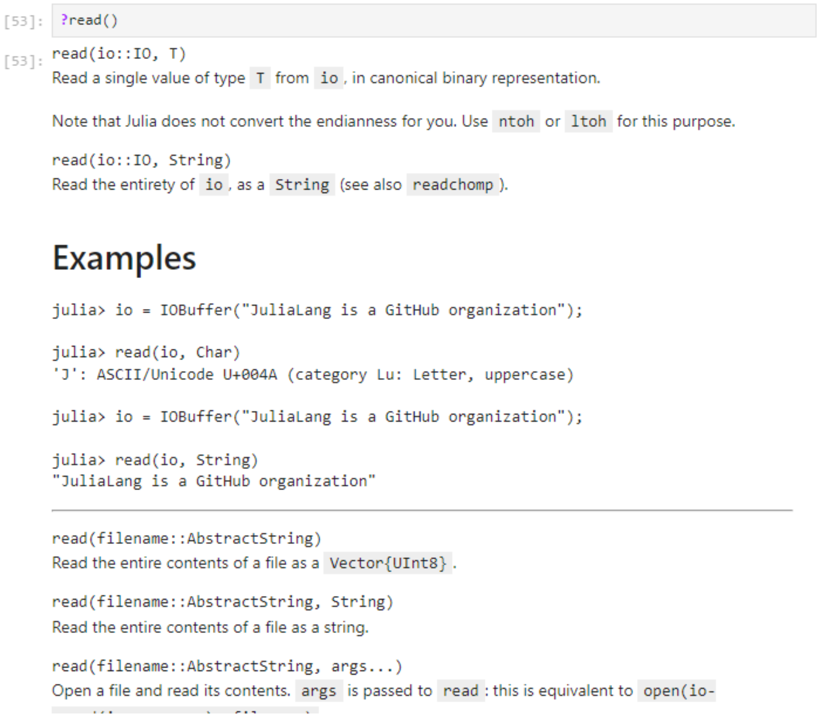
```
[43]: a = [4 7 6]
      b = [1, 2, 3]
      a[2], b[2]

[43]: (7, 2)

[46]: a = 1; b = 2; c = 3; d = 4
      Am = [a b; c d]

[46]: 2×2 Matrix{Int64}:
       1  2
       3  4

[47]: Am[1,1], Am[1,2], Am[2,1], Am[2,2]

[47]: (1, 2, 3, 4)

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

[49]: 1×1 Matrix{Int64}:
       27

[50]: aa, AA, aa'

[50]: ([1 2], [1 2; 3 4], [1; 2;;])
```

Рис. 3.14: Примеры работы с массивами

3.4 Самостоятельная работа

1. Изучила документацию по основным функциям Julia для чтения / записи / вывода информации на экран и привела свои примеры их использования:

- read() (рис. 3.15 и 3.16);



```
[53]: ?read()
```

```
[53]: 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`).

Examples

```
julia> io = IOBuffer("JuliaLang is a GitHub organization");

julia> read(io, Char)
'J': ASCII/Unicode U+004A (category Lu: Letter, uppercase)

julia> io = IOBuffer("JuliaLang is a GitHub organization");

julia> read(io, String)
"JuliaLang is a GitHub organization"
```

```
read(filename::AbstractString)
```

Read the entire contents of a file as a `Vector{UInt8}`.

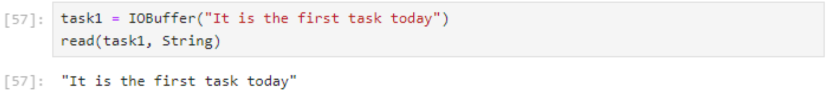
```
read(filename::AbstractString, String)
```

Read the entire contents of a file as a string.

```
read(filename::AbstractString, args...)
```

Open a file and read its contents. `args` is passed to `read`: this is equivalent to `open(io-`

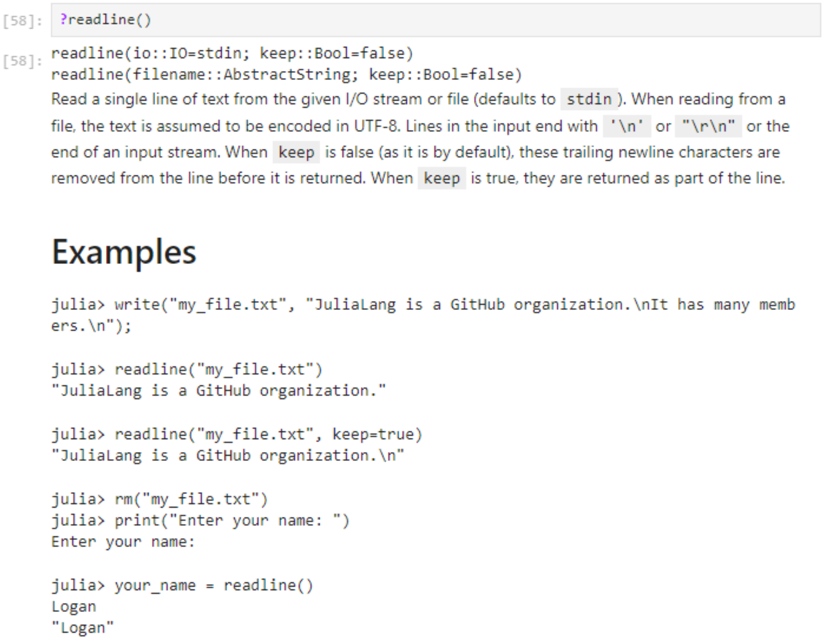
Рис. 3.15: Функция read(). Информация



```
[57]: task1 = IOBuffer("It is the first task today")
      read(task1, String)

[57]: "It is the first task today"
```

Рис. 3.16: Функция read(). Пример

- readline() (рис. 3.17 и 3.18);



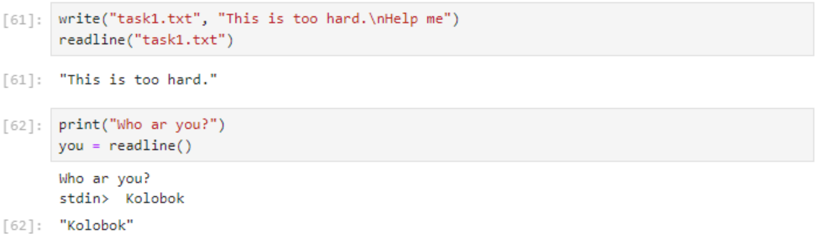
```
[58]: ?readline()

[58]: 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.

Examples

```
julia> write("my_file.txt", "JuliaLang is a GitHub organization.\nIt has many memb
ers.\n");

julia> readline("my_file.txt")
"JuliaLang is a GitHub organization."

julia> readline("my_file.txt", keep=true)
"JuliaLang is a GitHub organization.\n"

julia> rm("my_file.txt")
julia> print("Enter your name: ")
Enter your name:

julia> your_name = readline()
Logan
"Logan"
```

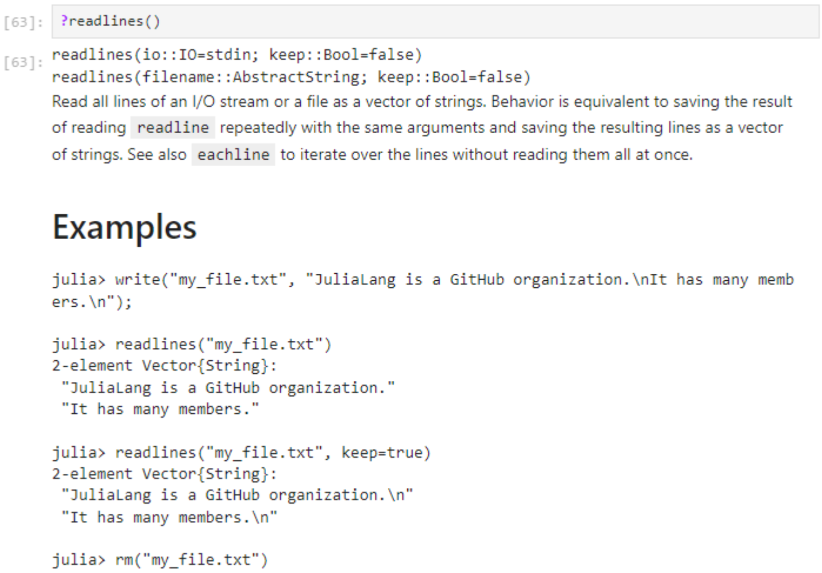
Рис. 3.17: Функция readline(). Информация



```
[61]: write("task1.txt", "This is too hard.\nHelp me")
      readline("task1.txt")

[61]: "This is too hard."

[62]: print("Who ar you?")
      you = readline()

      Who ar you?
      stdin>  Kolobok
[62]: "Kolobok"
```

Рис. 3.18: Функция readline(). Пример

- readlines() (рис. 3.19 и 3.20);



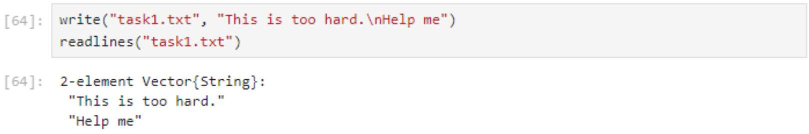
```
[63]: ?readlines()

[63]: 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.

Examples

```
julia> write("my_file.txt", "JuliaLang is a GitHub organization.\nIt has many memb
ers.\n");

julia> readlines("my_file.txt")
2-element Vector{String}:
 "JuliaLang is a GitHub organization."
 "It has many members."

julia> readlines("my_file.txt", keep=true)
2-element Vector{String}:
 "JuliaLang is a GitHub organization.\n"
 "It has many members.\n"

julia> rm("my_file.txt")
```

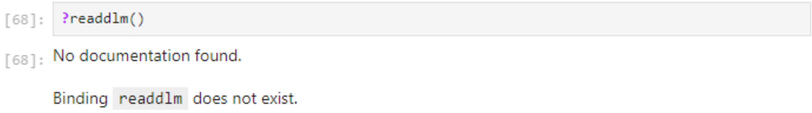
Рис. 3.19: Функция readlines(). Информация



```
[64]: write("task1.txt", "This is too hard.\nHelp me")
      readlines("task1.txt")

[64]: 2-element Vector{String}:
       "This is too hard."
       "Help me"
```

Рис. 3.20: Функция readlines(). Пример

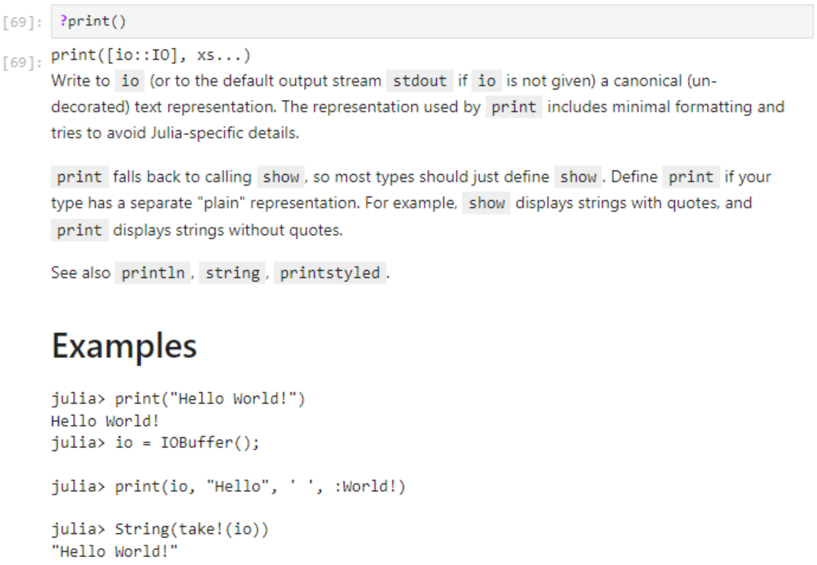
- readdlm() (рис. 3.21). Здесь информация не была предоставлена, потому и нет примера;



```
[68]: ?readdlm()

[68]: No documentation found.

      Binding readdlm does not exist.
```

Рис. 3.21: Функция readdlm(). Информация не получена

- print() (рис. 3.22 и 3.23);



```
[69]: ?print()
```

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

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

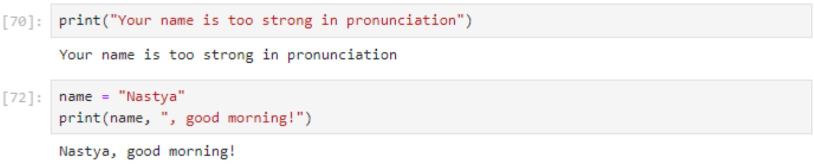
`print` falls back to calling `show`, so most types should just define `show`. Define `print` if your type has a separate "plain" representation. For example, `show` displays strings with quotes, and `print` displays strings without quotes.

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

Examples

```
julia> print("Hello World!")
Hello World!
julia> io = IOBuffer();

julia> print(io, "Hello", ' ', :World!)

julia> String(take!(io))
"Hello World!"
```

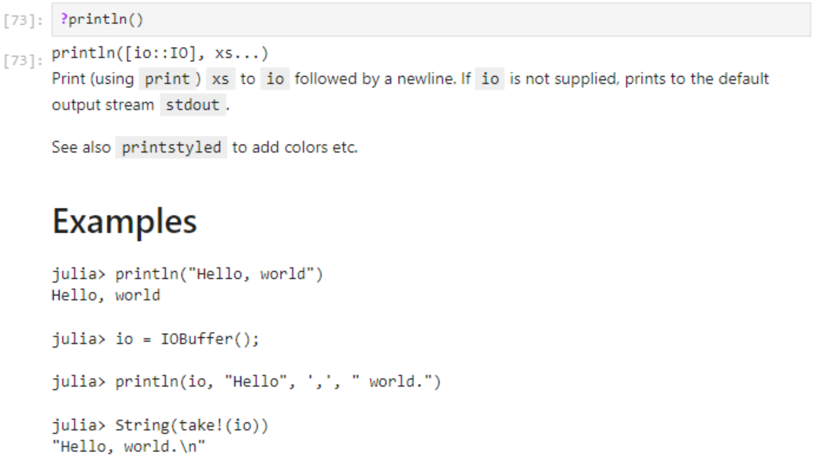
Рис. 3.22: Функция print(). Информация



```
[70]: print("Your name is too strong in pronunciation")

      Your name is too strong in pronunciation

[72]: name = "Nastya"
      print(name, ", good morning!")

      Nastya, good morning!
```

Рис. 3.23: Функция print(). Пример

- println() (рис. 3.24 и 3.25);



```
[73]: ?println()
```

```
[73]: println([io::IO], xs...)
```

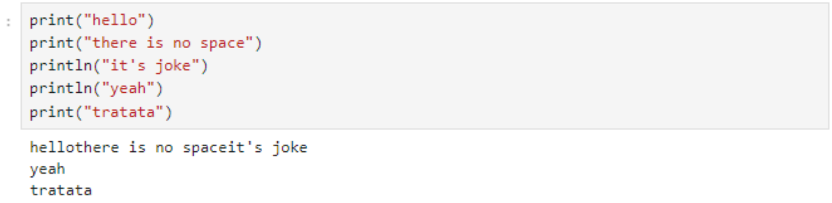
Print (using `print`) `xs` to `io` followed by a newline. If `io` is not supplied, prints to the default output stream `stdout`.

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. 3.24: Функция println(). Информация

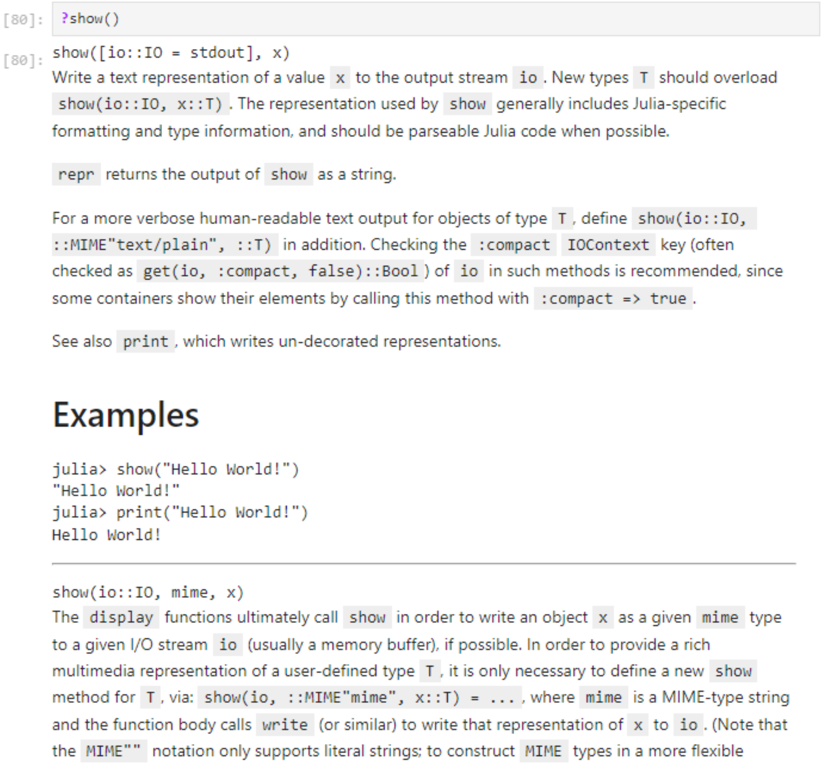


```
print("hello")
print("there is no space")
println("it's joke")
println("yeah")
print("tratata")
```

```
hellothere is no spaceit's joke
yeah
tratata
```

Рис. 3.25: Функция println(). Пример

- show() (рис. 3.26 и 3.27);



```
[80]: ?show()
```

```
[80]: show([io::IO = stdout], x)
```

Write a text representation of a value `x` to the output stream `io`. New types `T` should overload `show(io::IO, x::T)`. The representation used by `show` generally includes Julia-specific formatting and type information, and should be parseable Julia code when possible.

`repr` returns the output of `show` as a string.

For a more verbose human-readable text output for objects of type `T`, define `show(io::IO, ::MIME"text/plain", ::T)` in addition. Checking the `:compact` `IOContext` key (often checked as `get(io, :compact, false)::Bool`) of `io` in such methods is recommended, since some containers show their elements by calling this method with `:compact => true`.

See also `print`, which writes un-decorated representations.

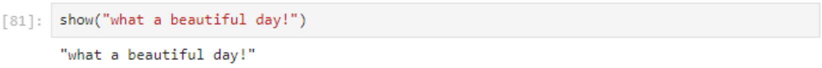
Examples

```
julia> show("Hello World!")
"Hello World!"
julia> print("Hello World!")
Hello World!
```

```
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

Рис. 3.26: Функция show(). Информация

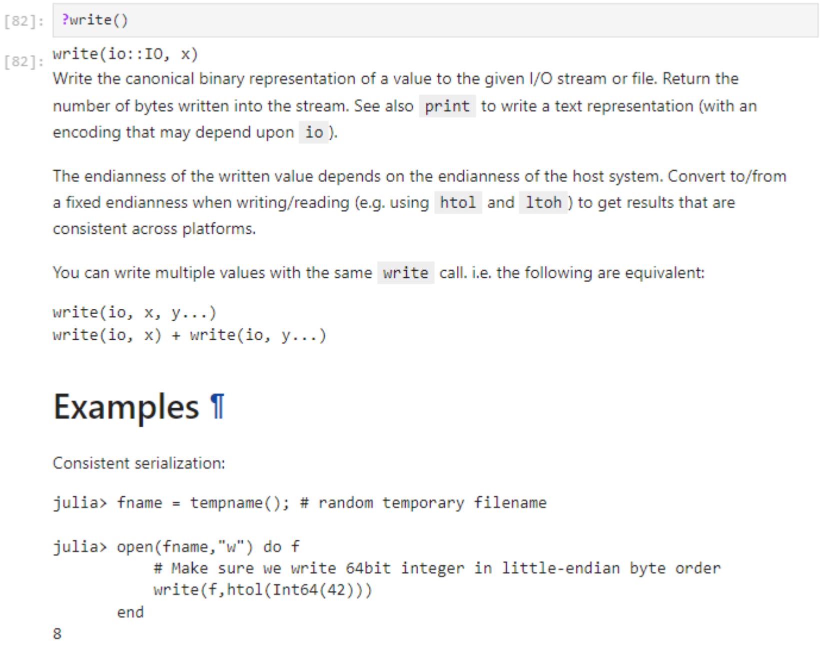


```
[81]: show("what a beautiful day!")
```

```
"what a beautiful day!"
```

Рис. 3.27: Функция show(). Пример

- write() (рис. 3.28 и 3.29).



```
[82]: ?write()
```

```
[82]: write(io::IO, 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:

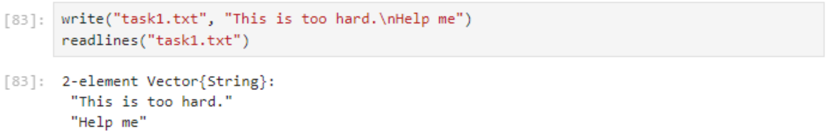
```
write(io, x, y...)
write(io, x) + write(io, y...)
```

Examples ¶

Consistent serialization:

```
julia> fname = tempname(); # random temporary filename

julia> open(fname,"w") do f
           # Make sure we write 64bit integer in little-endian byte order
           write(f,htol(Int64(42)))
       end
8
```

Рис. 3.28: Функция write(). Информация

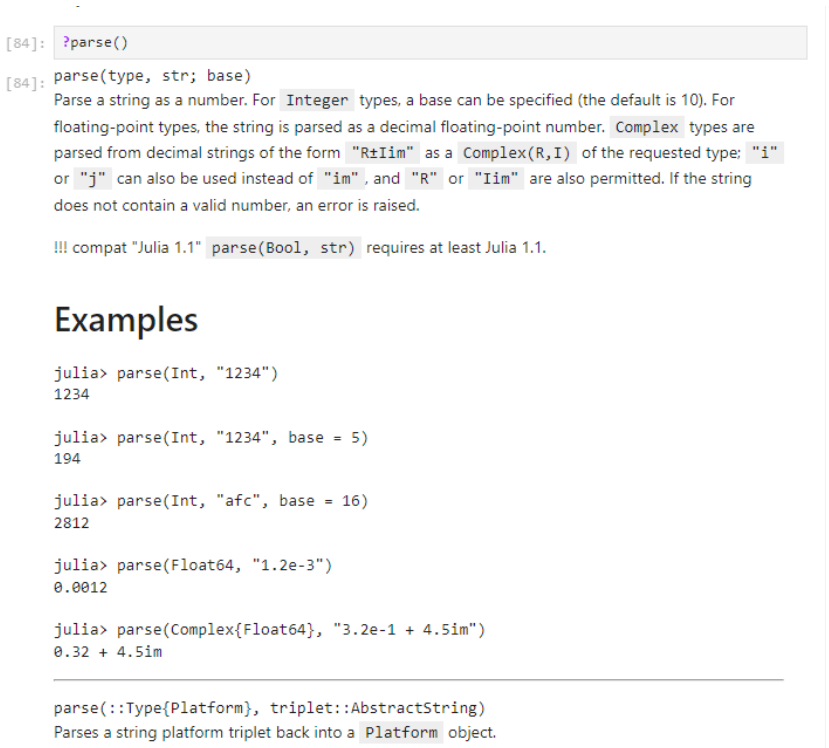


```
[83]: write("task1.txt", "This is too hard.\nHelp me")
      readlines("task1.txt")
```

```
[83]: 2-element Vector{String}:
       "This is too hard."
       "Help me"
```

Рис. 3.29: Функция write(). Пример

2. Изучила документацию по функции parse() (рис. 3.30). Приведила свои примеры её использования. На рис. 3.31 видно, что тип строки не получается конвертировать, потому сначала его нужна спарсить.



```
[84]: ?parse()
```

```
[84]: 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.

!!! compat "Julia 1.1" `parse(Bool, str)` requires at least Julia 1.1.

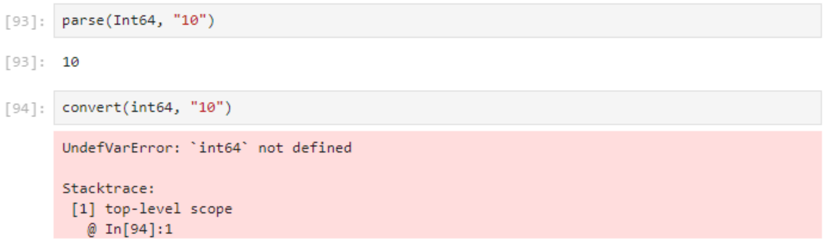
Examples

```
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
```

```
parse(::Type{Platform}, triplet::AbstractString)
```

Parses a string platform triplet back into a `Platform` object.

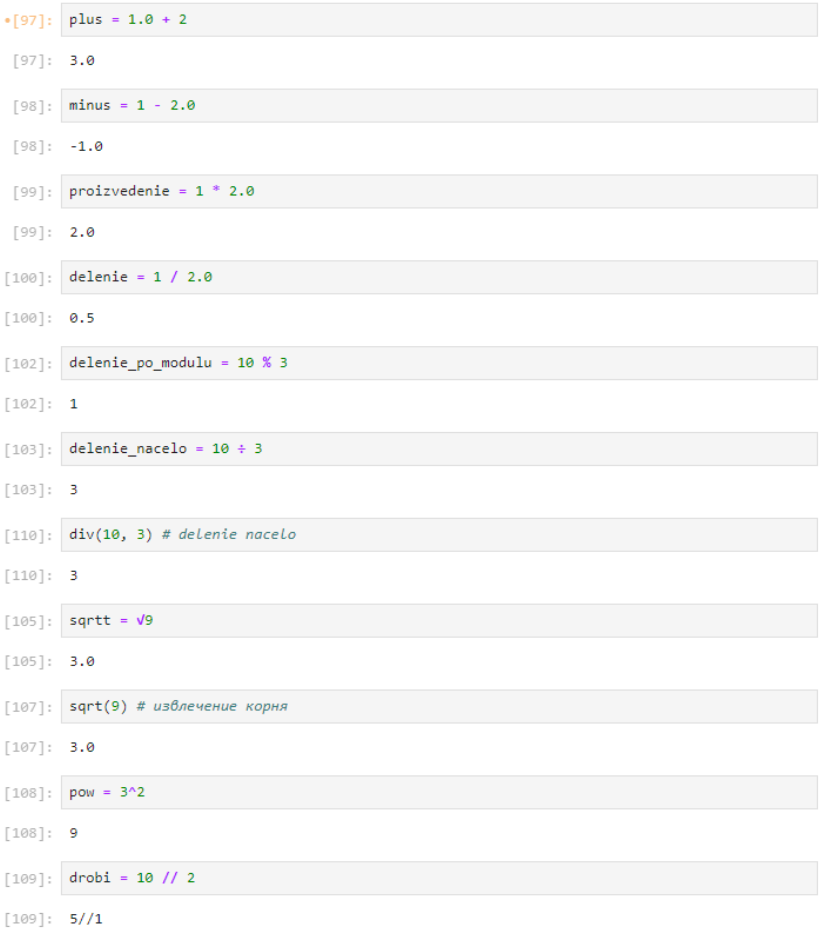
Рис. 3.30: Функция parse(). Информация



```
[93]: parse(Int64, "10")

[93]: 10

[94]: convert(int64, "10")

UndefVarError: `int64` not defined

Stacktrace:
 [1] top-level scope
   @ In[94]:1
```

Рис. 3.31: Функция parse(). Пример

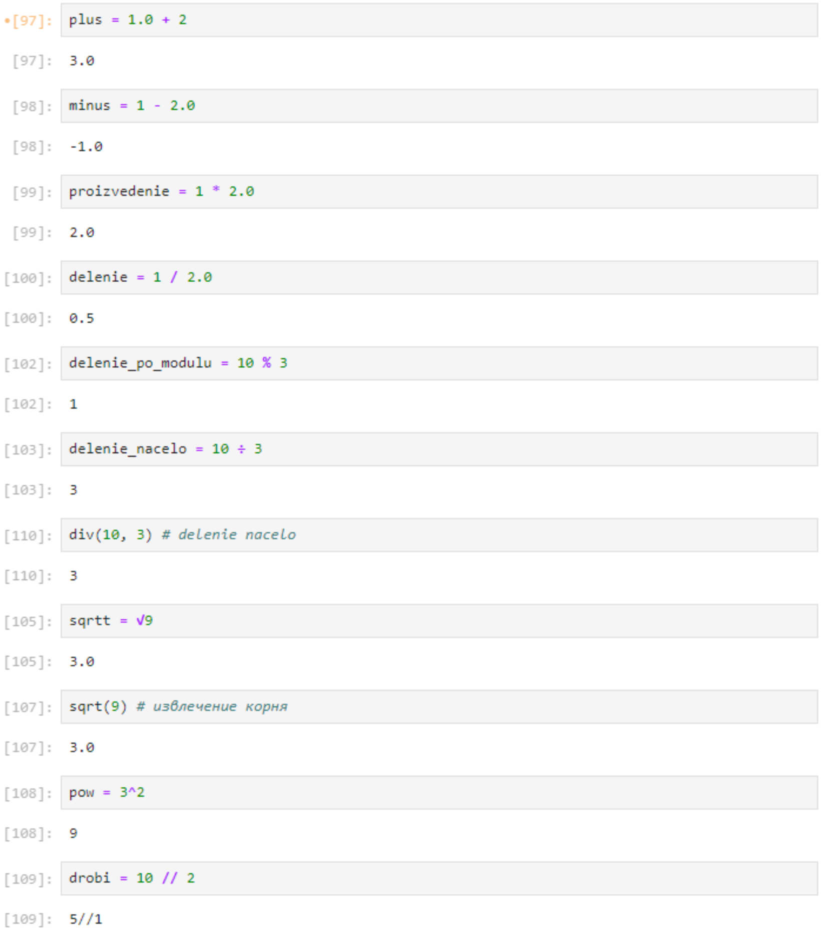
3. Изучила синтаксис Julia для базовых математических операций с разным типом переменных: сложение, вычитание, умножение, деление, возведение в степень, извлечение корня, сравнение, логические операции. Привела свои примеры с пояснениями по особенностям их применения (рис. 3.32).



```
•[97]:  plus = 1.0 + 2
 [97]:  3.0
 [98]:  minus = 1 - 2.0
 [98]:  -1.0
 [99]:  proizvedenie = 1 * 2.0
 [99]:  2.0
[100]:  delenie = 1 / 2.0
[100]:  0.5
[102]:  delenie_po_modulu = 10 % 3
[102]:  1
[103]:  delenie_nacelo = 10 ÷ 3
[103]:  3
[110]:  div(10, 3) # delenie nacelo
[110]:  3
[105]:  sqrtt = √9
[105]:  3.0
[107]:  sqrt(9) # извлечение корня
[107]:  3.0
[108]:  pow = 3^2
[108]:  9
[109]:  drobi = 10 // 2
[109]:  5//1
```

Рис. 3.32: Примеры для базовых математических операций

4. Привела несколько своих примеров с пояснениями с операциями над матрицами и векторами: сложение, вычитание, скалярное произведение, транспонирование, умножение на скаляр (рис. 3.33).



```
•[97]: plus = 1.0 + 2

 [97]: 3.0

 [98]: minus = 1 - 2.0

 [98]: -1.0

 [99]: proizvedenie = 1 * 2.0

 [99]: 2.0

[100]: delenie = 1 / 2.0

[100]: 0.5

[102]: delenie_po_modulu = 10 % 3

[102]: 1

[103]: delenie_nacelo = 10 ÷ 3

[103]: 3

[110]: div(10, 3) # delenie nacelo

[110]: 3

[105]: sqrtt = √9

[105]: 3.0

[107]: sqrt(9) # извлечение корня

[107]: 3.0

[108]: pow = 3^2

[108]: 9

[109]: drobi = 10 // 2

[109]: 5//1
```

Рис. 3.33: Примеры с операциями над матрицами

4 Выводы

В ходе лабораторной работы мной было подготовлено рабочее пространство и инструментарий для работы с языком программирования Julia, также я познакомилась с основами синтаксиса Julia на простейших примерах.