

Лабораторная работа №1

Julia. Установка и настройка. Основные принципы

Коняева Марина Александровна

НФИбд-01-21

Студ. билет: 1032217044

2024

RUDN

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

Julia — высокоуровневый свободный язык программирования с динамической типизацией, созданный для математических вычислений. Эффективен также и для написания программ общего назначения[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).

Устанавливаем инструментарий под ОС Windows: См. рис. 1, См. рис. 2, См. рис. 3? См. рис. 4, См. рис. 5, См. рис. 6

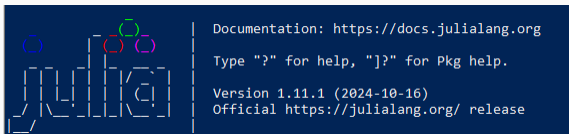


Рис. 1: Установка Julia (уже установлена)

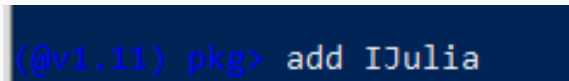


Рис. 2: Установка IJulia

```
PS C:\Users\user> Set-ExecutionPolicy Bypass -Scope Process -Force; [System.Net.ServicePointManager]::SecurityProtocol =  
[System.Net.ServicePointManager]::SecurityProtocol -bor 3072; $ex ((New-Object System.Net.WebClient).DownloadString("ht  
tp://community.chocolatey.org/install.ps1"))
```

Рис. 3: Установка chocolatey

```
Ensuring chocolatey mapping is in the 118-102  
PS C:\Users\user> choco install far -y
```

Рис. 4: Установка пакета far

```
See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log)  
PS C:\Users\user> choco install notepadplusplus -y
```

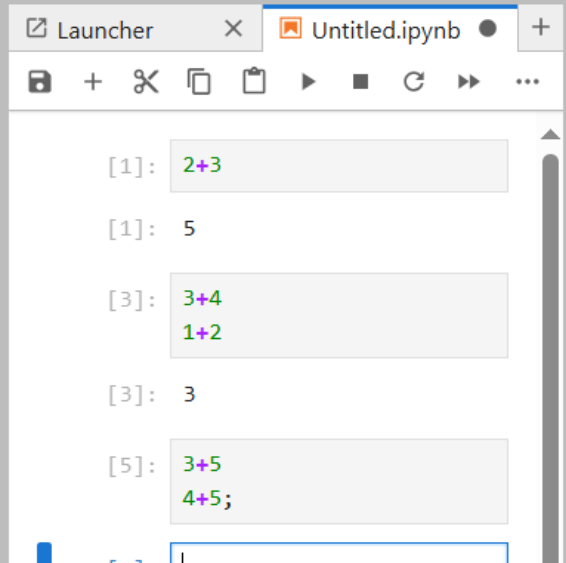
Рис. 5: Установка notepad++

```
See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log)  
PS C:\Users\user> choco install anaconda3 -y
```

Рис. 6: Установка anaconda3

Выполнение лабораторной работы

Выполняем примеры из раздела 1.3.3: См. рис. 7, См. рис. 8, См. рис. 9, См. рис. 10, См. рис. 11, См. рис. 12



The screenshot shows a Jupyter Notebook window with two tabs: 'Launcher' and 'Untitled.ipynb'. The 'Untitled.ipynb' tab is active, displaying a notebook with four code cells. The first cell contains the expression `2+3` and the output `[1]: 5`. The second cell contains the expressions `3+4` and `1+2` on separate lines, with the output `[3]: 3`. The third cell contains the expressions `3+5` and `4+5;` on separate lines. The fourth cell is partially visible at the bottom, showing the expression `5+3`.

```
[1]: 2+3
```

```
[1]: 5
```

```
[3]: 3+4
     1+2
```

```
[3]: 3
```

```
[5]: 3+5
     4+5;
```

```
[5]: 5+3
```

```
[7]: ?println
```

```
search: println print sprint pointer p  
rintstyled
```

```
[7]: 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"
```


Выполнение лабораторной работы

```
[11]: typeof(3), typeof(3.5), typeof(3/3.55), typeof(sqrt(3+4im)), typeof(pi)

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

[13]: 1.0/0.0, 1.0/(-0.0), 0.0/0.0

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

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

[15]: (Float64, Float64, Float64)

[17]: 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. 9: Определение типов переменных

```
[20]: Int64(2.0), Char(2), typeof(Char(2))  
[20]: (2, '\x02', Char)  
[22]: convert{Int64, 2.0}, convert{Char, 2}  
[22]: (2, '\x02')  
[24]: typeof(promote{Int8(1), Float16(4.5), Float32(4.1)})  
[24]: Tuple{Float32, Float32, Float32}
```

Рис. 10: Определение типов переменных_2

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

[26]: f (generic function with 1 method)

[28]: f(4)

[28]: 16

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

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

[32]: g(8)

[32]: 64
```

Рис. 11: Работа с функциями

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

[34]: (7, 2)

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

[36]: 2x2 Matrix{Int64}:
      1  2
      3  4

[38]: Am[1,1], Am[1,2], Am[2,1], Am[2,2]

[38]: (1, 2, 3, 4)

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

[40]: 1x1 Matrix{Int64}:
      27

[42]: aa, AA, aa'

[42]: ([1 2], [1 2; 3 4], [1; 2;])
```

Рис. 12: Работа с векторами и массивами

Задание для самостоятельного выполнения

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

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

Выполнение лабораторной работы

Познакомились со справками к некоторым функциям (в отчете):

Приведем примеры использования данных функций: См. рис. 13

```
[ ]: content = read("example.txt", String)
      println(content)
```

```
[ ]: file = open("example.txt", "r")
      line = readline(file)
      println(line)
      close(file)
```

```
[ ]: lines = readlines("example.txt")
      println(lines)
```

```
[ ]: using DelimitedFiles
      data = readdlm("data.csv", ',')
      println(data)
```

```
[ ]: print("Hello, World!")
```

```
[ ]: println("Hello, World!")
```

```
[ ]: show(3.34159)
```

```
[ ]: file = open("example.txt", "w")
      write(file, "Hello, World!")
      close(file)
```

```
[ ]: num = parse{Int, "123"}
      println(num)
```



Рис. 13: Примеры использования функций

Пояснения к ним:

- `read()`: Читает весь контент из файла или потока. Используется для чтения данных целиком из файла или потока в виде строки или байтов.
- `readline()`: Читает одну строку из файла или потока. Используется для построчного чтения данных.
- `readlines()`: Читает все строки из файла или потока в виде массива строк. Полезно, если нужно сразу получить доступ ко всем строкам файла.

- `readdelim()`: Читает данные из файла, разделенные разделителями, и возвращает массив. Используется для работы с табличными данными, такими как CSV-файлы.
- `print()`: Выводит текст без переноса строки. Полезно для вывода текста на одной строке.
- `println()`: Выводит текст с переносом строки. Используется для вывода текста с автоматическим переходом на следующую строку.

- `show()`: Выводит объект в более техническом формате. Предназначен для отображения данных в формате, удобном для разработчиков.
- `write()`: Записывает данные в файл или поток. Используется для записи строк или байтов в файл.
- `parse()`: Преобразует строку в заданный тип данных. Удобен для конвертации строк в числа или другие типы, такие как `Float64`.

Приведем примеры базовых математических операций: См. рис. 14

```
[75]: res = 3+5  
      print(res)  
      8  
  
[77]: res = 3-5  
      print(res)  
      -2  
  
[79]: res = 3*5  
      print(res)  
      15  
  
[81]: res = 15 / 3  
      print(res)  
      5.0  
  
[83]: res = 2**5  
      print(res)  
      32  
  
[85]: res = sqrt(16)  
      print(res)  
      4.0  
  
[87]: res = 5>3  
      print(res)  
      true  
  
[89]: res = true && false  
      print(res)  
      false
```

Рис. 14: Примеры использования математических операций

Приведем примеры операций с матрицами и векторами: См. рис. 15

```
[91]: vec1 = [1, 2, 3]
      vec2 = [4, 5, 6]
      res = vec1 + vec2
      print(res)

      [5, 7, 9]

[93]: vec1 = [1, 2, 3]
      vec2 = [4, 5, 6]
      res = vec1 - vec2
      print(res)

      [-3, -3, -3]

[97]: matrix = [1 2 3; 4 5 6]
      res = transpose(matrix)
      print(res)

      [1 4; 2 5; 3 6]

[99]: vec1 = [1,2,3]
      scalar = 2
      res = scalar * vec1
      print(res)

      [2, 4, 6]

[103]: using LinearAlgebra

      v1 = [1, 2, 3]
      v2 = [4, 5, 6]
      result = dot(v1, v2)
      println(result)

      32
```

Рис. 15: Примеры использования математических операций с матрицами и векторами

Пояснения к ним:

- Сложение: Векторы складываются поэлементно.
- Вычитание: Вычитание выполняется поэлементно.
- Скалярное произведение: Используется функция `dot()` из библиотеки `LinearAlgebra` для вычисления суммы произведений соответствующих элементов двух векторов.
- Транспонирование: Функция `transpose()` возвращает транспонированную матрицу.
- Умножение на скаляр: Умножение вектора или матрицы на число выполняется поэлементно.

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

[1] Julia: <https://ru.wikipedia.org/wiki/Julia>