Лабораторная работа №1

Компьютерный практикум по статистическому анализу данных

Николаев Дмитрий Иванович

Российский университет дружбы народов, Москва, Россия

Прагматика выполнения

Прагматика выполнения

- Получение навыков работы в Jupyter Notebook
- Освоение базовых особенностей языка Julia

Цель

Цель

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

Задачи

Задачи

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).

Выполнение работы

Повторение примеров из пункта 1.3.3 (1/3)

Задание №1.3.3

Ввод [1]:
```
typeof(3), typeof(0.5), typeof(1 + 2im), typeof(√2), typeof("a"), typeof('a'), typeof(π)
```
Out[1]:
```
(Int64, Float64, Complex{Int64}, Float64, String, Char, Irrational{:π})
```

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

Ввод [3]:
```
typeof(1.0/0.0), typeof(1.0/(-0.0)), typeof(0.0/0.0)
```
Out[3]:
```
(Float64, Float64, Float64)
```

Ввод [4]:
```
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]
```

Ввод [5]:
```
Int64(2.0), Char(2), typeof(Char(2))
```
Out[5]:
```
(2, '\x02', Char)
```

Рис. 1: Работа с типами

Повторение примеров из пункта 1.3.3 (2/3)

```
Ввод [6]: convert(Int64, 2.0), convert(Char,2)
Out[6]: (2, '\x02')

Ввод [7]: typeof(promote(Int8(1), Float16(4.5), Float32(4.1)))
Out[7]: Tuple{Float32, Float32, Float32}

Ввод [8]: function f(x)
              x^2
          end
Out[8]: f (generic function with 1 method)

Ввод [9]: f(4)
Out[9]: 16

Ввод [10]: g(x)=x^2
Out[10]: g (generic function with 1 method)

Ввод [11]: g(4)
Out[11]: 16

Ввод [13]: a = [4 7 6]
           b = [1, 2, 3];

Ввод [14]: a[2]
Out[14]: 7

Ввод [15]: b[1]
```

Повторение примеров из пункта 1.3.3 (3/3)

```
Ввод [15]: b[1]

Out[15]: 1

Ввод [16]: 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

Out[16]: (1, 2, 3, 4)

Ввод [17]: aa = [1 2]
           AA = [1 2; 3 4]
           aa*AA*aa'

Out[17]: 1×1 Matrix{Int64}:
          27

Ввод [18]: aa, AA, aa'

Out[18]: ([1 2], [1 2; 3 4], [1; 2;;])
```

Рис. 3: Векторы и матрицы

Команда read (1/2)

Побитовое представление символов

Ввод [25]:
```
input = IOBuffer("ahjaghjkewhjtwekrj")
read(input)
```

Out[25]:
```
18-element Vector{UInt8}:
 0x61
 0x68
 0x6a
 0x61
 0x67
 0x68
 0x6a
 0x6b
 0x65
 0x77
 0x68
 0x6a
 0x74
 0x77
 0x65
 0x6b
 0x72
 0x6a
```

Чтение первого символа

Ввод [32]:
```
input = IOBuffer("ahjaghjkewhjtwekrj")
read(input, Char)
```

Out[32]:
```
'a': ASCII/Unicode U+0061 (category Ll: Letter, lowercase)
```

Команда read (2/2)

Рис. 5: Чтение одной строки целиком и чтение конкретного числа символов

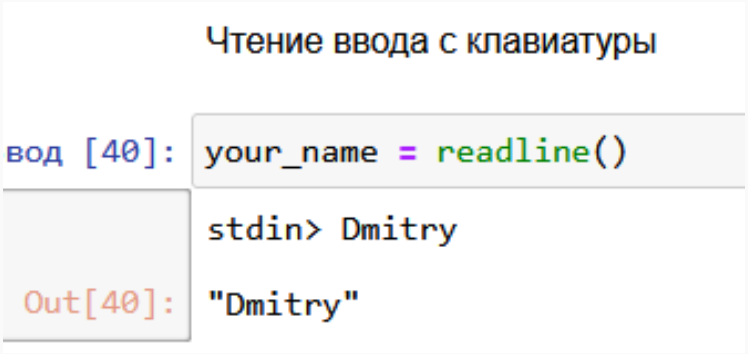


Рис. 6: Чтение с клавиатуры

Чтение из файла, с несколькими строками данных

```
Ввод [49]:
open("my_file.txt", "w") do io
    write(io, "My name is Dmitry.\nI'm 21.\n");
end
your_info = readlines("my_file.txt")

Out[49]:
2-element Vector{String}:
 "My name is Dmitry."
 "I'm 21."

Ввод [50]:
rm("my_file.txt")
```

Рис. 7: Чтение строк из файла

Команда readdir

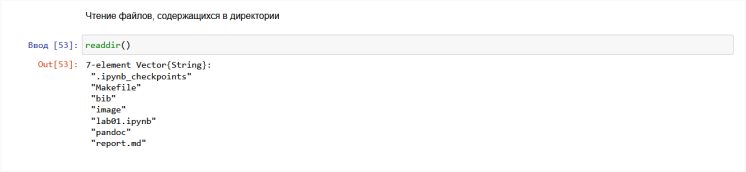


Рис. 8: Считывание файлов из рабочей директории

Команда print

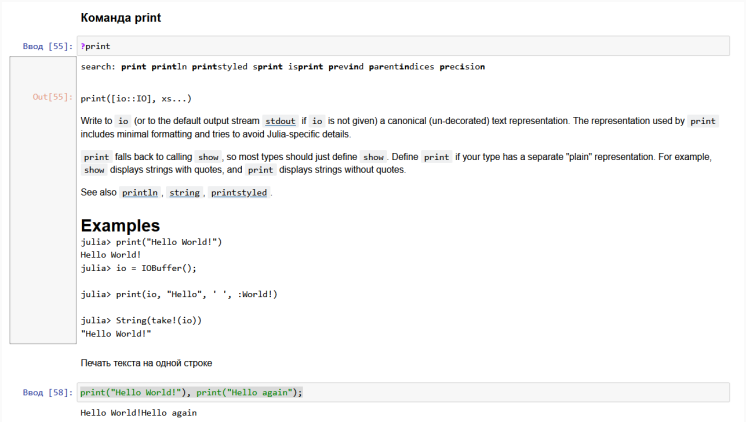
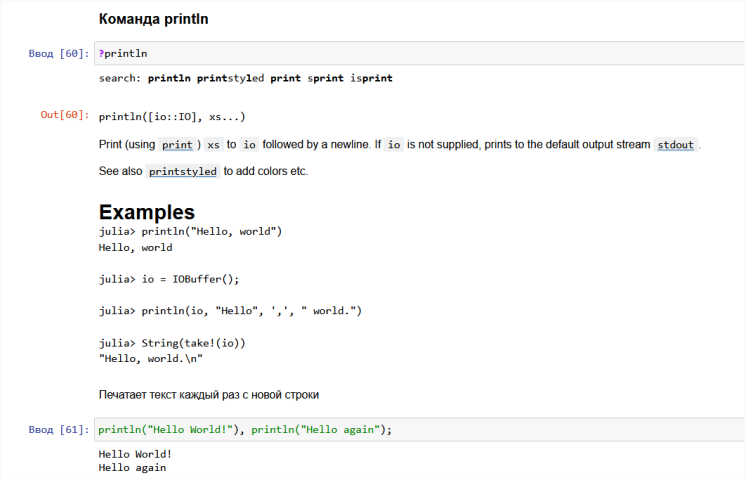


Рис. 9: Справка команды print и проверка ее рабоыт

Команда println



Команда println

Ввод [60]: `?println`

```
search: println printstyled print sprint isprint
```

Out[60]:

```
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"
```

Печатает текст каждый раз с новой строки

Ввод [61]:

```
println("Hello World!"), println("Hello again");
```

```
Hello World!
Hello again
```

Рис. 10: Справка команды println и проверка ее рабоыт

Команда show

Печатает текст с сохранением особенностей типа данных переданного аргумента

```
Ввод [66]: show("Hello World!"), show('a');
```

```
"Hello World!"'a'
```

Рис. 11: Печать с помощью команды show

Команда write

Записывает информация в файл или поток ввода/вывода

```
Ввод [68]: io = IOBuffer();
           write(io, "My name is Dmitry and", " I'm 21.")
 Out[68]: 29

Ввод [69]: String(take!(io))
 Out[69]: "My name is Dmitry and I'm 21."
```

Рис. 12: Запись в файл

Команда parse

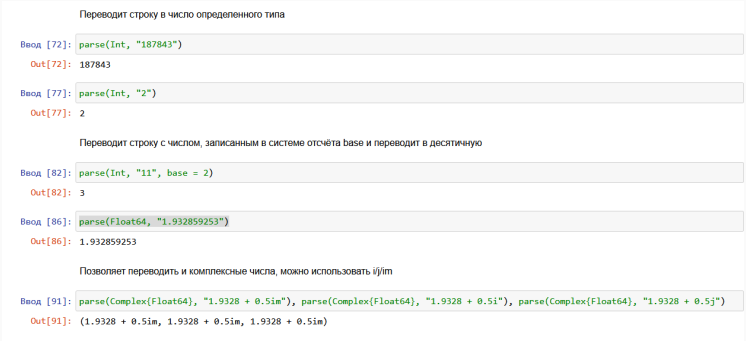
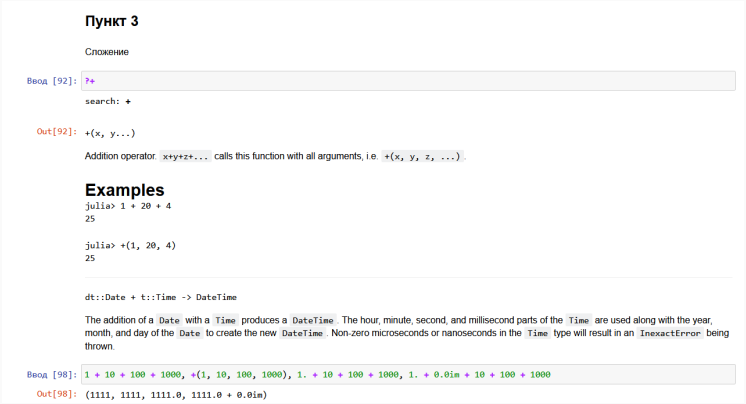


Рис. 13: Проверка работы команды parse

Оператор сложения



Пункт 3

Сложение

Ввод [92]:
```
?+
```

```
search: +
```

Out[92]:
```
+(x, y...)
```

Addition operator. `x+y+z+...` calls this function with all arguments, i.e. `+(x, y, z, ...)`.

Examples

```
julia> 1 + 20 + 4
25

julia> +(1, 20, 4)
25
```

```
dt::Date + t::Time -> DateTime
```

The addition of a `Date` with a `Time` produces a `DateTime`. The hour, minute, second, and millisecond parts of the `Time` are used along with the year, month, and day of the `Date` to create the new `DateTime`. Non-zero microseconds or nanoseconds in the `Time` type will result in an `InexactError` being thrown.

Ввод [98]:
```
1 + 10 + 100 + 1000, +(1, 10, 100, 1000), 1. + 10 + 100 + 1000, 1. + 0.0im + 10 + 100 + 1000
```

Out[98]:
```
(1111, 1111, 1111.0, 1111.0 + 0.0im)
```

Рис. 14: Проверка работы оператора сложения +

Операторы вычитания, умножения и деления

Вычитание

```
Ввод [ ]: ?-
```

```
Ввод [100]: 2 - 3, -(4, 3), 0 - 1, 0.1 - 0.3, 0.5 + 1.2im - 0.7im - 0.3
Out[100]: (-1, 1, -1, -0.19999999999999998, 0.2 + 0.5im)
```

Умножение

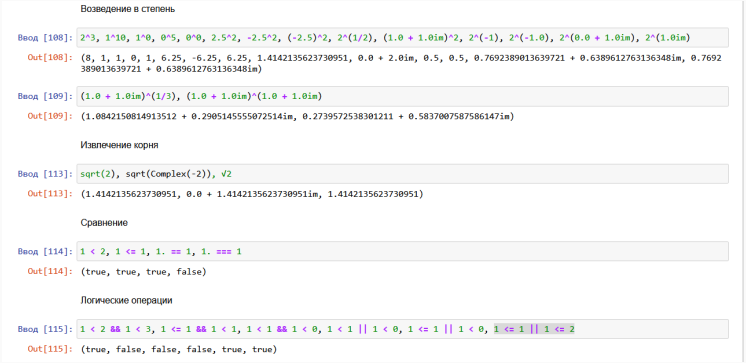
```
Ввод [102]: 1*2*3, -2*3, 2.5*4, (1.0 + 1.0im)*(1.0 - 1.0im)
Out[102]: (6, -6, 10.0, 2.0 + 0.0im)
```

Деление

```
Ввод [105]: 2/2, Int(2/2), 3/2, 2.5/2, 1/3, π/4, -π/4, 5//6, 10//12
Out[105]: (1.0, 1, 1.5, 1.25, 0.3333333333333333, 0.7853981633974483, -0.7853981633974483, 5//6, 5//6)
```

Рис. 15: Проверка работы операторов вычитания -, умножения * и деления /

Возведение в степень, извлечение корня, операторы сравнения и логические операции



Возведение в степень

```
Ввод [108]: 2^3, 1^10, 1^0, 0^5, 0^0, 2.5^2, -2.5^2, (-2.5)^2, 2^(1/2), (1.0 + 1.0im)^2, 2^(-1), 2^(-1.0), 2^(0.0 + 1.0im), 2^(1.0im)
Out[108]: (8, 1, 1, 0, 1, 6.25, -6.25, 6.25, 1.4142135623730951, 0.0 + 2.0im, 0.5, 0.5, 0.7692389013639721 + 0.6389612763136348im, 0.7692389013639721 + 0.6389612763136348im)

Ввод [109]: (1.0 + 1.0im)^(1/3), (1.0 + 1.0im)^(1.0 + 1.0im)
Out[109]: (1.0842150814913512 + 0.2905145555072514im, 0.2739572538301211 + 0.5837007587586147im)
```

Извлечение корня

```
Ввод [113]: sqrt(2), sqrt(Complex(-2)), √2
Out[113]: (1.4142135623730951, 0.0 + 1.4142135623730951im, 1.4142135623730951)
```

Сравнение

```
Ввод [114]: 1 < 2, 1 <= 1, 1. == 1, 1. === 1
Out[114]: (true, true, true, false)
```

Логические операции

```
Ввод [115]: 1 < 2 && 1 < 3, 1 <= 1 && 1 < 1, 1 < 1 && 1 < 0, 1 < 1 || 1 < 0, 1 <= 1 || 1 < 0, 1 <= 1 || 1 <= 2
Out[115]: (true, false, false, false, true, true)
```

Рис. 16: Проверка работы операций возведения в степень ^, извлечения корня sqrt, операторов сравнения и логических операций

Работа с векторами и матрицами (1/3)

Пункт 4

Ввод [122]:
```
a = [1 2]
b = a'
A = [1 2; 3 4]
a
```

Out[122]:
```
1×2 Matrix{Int64}:
 1  2
```

Ввод [123]:
```
b
```

Out[123]:
```
2×1 adjoint(::Matrix{Int64}) with eltype Int64:
 1
 2
```

Ввод [124]:
```
A
```

Out[124]:
```
2×2 Matrix{Int64}:
 1  2
 3  4
```

Ввод [128]:
```
a + b;  # Несоответствие размерностей
```

```
DimensionMismatch: dimensions must match: a has dims (Base.OneTo(1), Base.OneTo(2)), b has dims (Base.OneTo(2), Base.OneTo(1)),
mismatch at 1

Stacktrace:
 [1] promote_shape
   @ .\indices.jl:178 [inlined]
 [2] promote_shape(a::Matrix{Int64}, b::LinearAlgebra.Adjoint{Int64, Matrix{Int64}})
   @ Base .\indices.jl:169
 [3] +(A::Matrix{Int64}, B::LinearAlgebra.Adjoint{Int64, Matrix{Int64}})
   @ Base .\arraymath.jl:7
 [4] top-level scope
```

Рис. 17: Создание строки, столбца и матрицы

Работа с векторами и матрицами (2/3)

```
Ввод [129]: a + b'
Out[129]: 1×2 Matrix{Int64}:
           2  4

Ввод [130]: a * b
Out[130]: 1×1 Matrix{Int64}:
           5

Ввод [133]: sqrt(a*a') # Норма вектора
Out[133]: 1×1 Matrix{Float64}:
           2.2360679774979

Ввод [135]: A*A
Out[135]: 2×2 Matrix{Int64}:
           7  10
          15  22

Ввод [136]: 2*A
Out[136]: 2×2 Matrix{Int64}:
           2  4
           6  8
```

Рис. 18: Сложение векторов, скалярное произведение векторов, умножение на скаляр и умножение матриц

Работа с векторами и матрицами (3/3)

```
Ввод [137]: a*A
Out[137]: 1×2 Matrix{Int64}:
           7  10

Ввод [138]: A*b
Out[138]: 2×1 Matrix{Int64}:
           5
          11

Ввод [139]: a*A*b
Out[139]: 1×1 Matrix{Int64}:
          27

Ввод [140]: A - A
Out[140]: 2×2 Matrix{Int64}:
          0  0
          0  0
```

Рис. 19: Умножение строки на матрицу и матрицы на столбец

Результаты

Результаты

В ходе работы я освоил основы языка Julia: работа с командами ввода/вывода, простейшие математические операции, операторы сравнения и логические операции.