Лабораторная работа № 1

Julia. Установка и настройка. Основные принципы

Абд эль хай мохамад

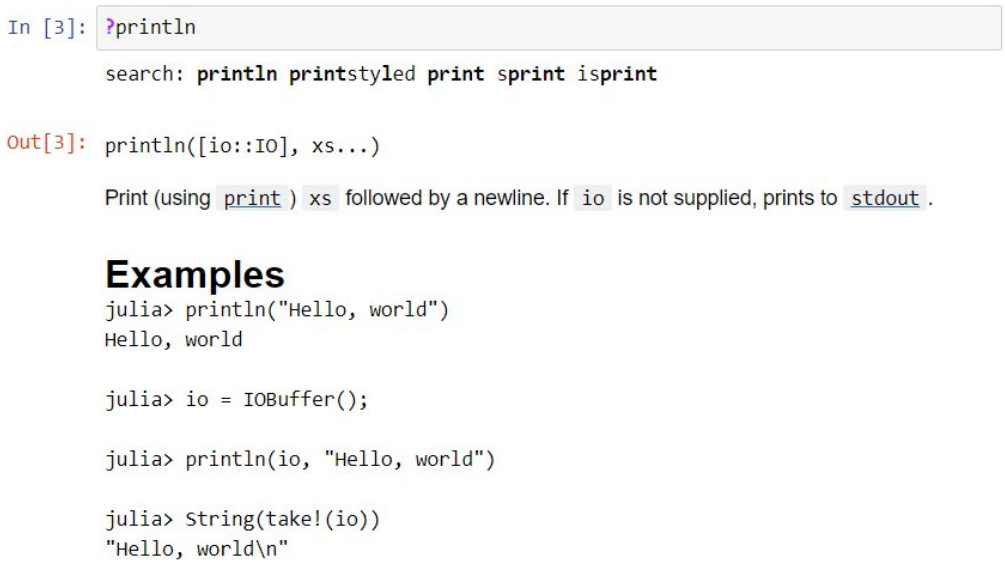
Содержание

1 . Цель работы

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

2 . Выполнение лабораторной работы

С помощью вопросительного знака получил информацию по работе функции println().



Основы синтаксиса Julia на примерах.

Далее я рассмотрю приведенные в задании примеры в своем ноутбуке.

```
In [4]: typeof(3), typeof(3.5), typeof(3/3.55), typeof(sqrt(3+4im)), typeof(pi)
Out[4]: (Int64, Float64, Float64, Complex{Float64}, Irrational{:π})

In [5]: 1.0/0.0, 1.0/(-0.0), 0.0/0.0
Out[5]: (Inf, -Inf, NaN)

In [6]: typeof(1.0/0.0), typeof(1.0/-0.0), typeof(0.0/0.0)
Out[6]: (Float64, Float64, Float64)

In [8]: for T in [Int8, Int16, Int32, Int64, Int128, 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]
 UInt16: [0, 65535]
 UInt32: [0, 4294967295]
 UInt64: [0, 18446744073709551615]
UInt128: [0, 340282366920938463463374607431768211455]
```

Рисунок 1. Примеры определения типа числовых величин

Ниже рассматриваются разные способы для преобразования типов данных на языке Julia.

```
In [9]: # преобразование прямым указанием
        Int64(2.0), Char(2), typeof(Char(2))
Out[9]: (2, '\x02', Char)

In [10]: # преобразование обощенным оператором
         convert(Int64, 2.0), convert(Char, 2)
Out[10]: (2, '\x02')

In [12]: # преобразование нескольких аргументов к одному типу
         typeof(promote(Int8(1), Float16(4.5), Float32(4.1)))
Out[12]: Tuple{Float32,Float32,Float32}
```

Рисунок 2. Примеры приведения аргументов к одному типу

Базовый синтаксис определения функции.

function <Имя> (<Список Параметров>)

<Действия>

End

```
In [11]: function f(x)
             x^2
         end

Out[11]: f (generic function with 1 method)

In [12]: f(4)

Out[12]: 16

In [13]: g(x) = x^2

Out[13]: g (generic function with 1 method)

In [14]: g(8)

Out[14]: 64
```

Рисунок 3. Пример определения функций.

```
In [15]: a = [4 7 6] # вектор-строка
         b = [1, 2, 3] #вектор-столбец
         a[2], b[2] # получим вторые элементы векторов a и b

Out[15]: (7, 2)

In [16]: a = 1; b = 2; c = 3; d = 4 # присвоение значений
         Am = [a b; c d] # матрица размером 2x2

Out[16]: 2×2 Array{Int64,2}:
          1  2
          3  4

In [17]: Am[1, 1], Am[1, 2], Am[2, 1], Am[2, 2] # элементы матрицы Am

Out[17]: (1, 2, 3, 4)

In [18]: aa = [1 2] #вектор-строка
         AA = [1 2; 3 4] #матрица 2x2
         aa*AA*aa' # умножение вектора-строки на матрицу и на вектор-столбец (операция

Out[18]: 1×1 Array{Int64,2}:
          27

In [19]: aa, AA, aa'

Out[19]: ([1 2], [1 2; 3 4], [1; 2])
```

Рисунок 4. Пример работы с массивами

Задания для самостоятельной работы.

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.

Для каждой функции в первой ячейке я вывожу информацию о ней (с помощью вопросительного знака). А во второй привожу пример использования.

Начнем с функции read(). Данная функция считывает значение типа T из io, в общепринятом двоичном представлении.

```
?read()
```

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

Пример: Выведем строку “Hello world”.

```
# попробуем вывести строку hello world
str_hello = IOBuffer("Hello world!");
read(str_hello, String)
```

```
"Hello world!"
```

Функция readline(). Считывание одной строки текста из данного потока ввода-вывода или файла. Строки на входе в файле заканчиваются буквой "\n " или "\r\n " или концом входного потока. Если keep имеет значение false (по умолчанию), эти конечные символы новой строки удаляются из строки до ее возврата. Когда keep имеет значение true, они возвращаются как часть строки.

```
In [23]: open("my_file.txt", "w") do io
             write(io, "Hello World and Julia!\n");
         end
```

```
Out[23]: 23
```

Как мы видим в начале я открыл текстовый файл на запись и поместил в него строку “Hello world and Julia!” и в конце указал ‘\n’. Функция write выводит нам на количество байтов, записанных в поток.. Далее с помощью функции readline() я попробовал 2 способа вывода записи, без параметра keep и с ним. В итоге я получил следующее:

```
In [24]: readline("my_file.txt")
Out[24]: "Hello World and Julia!"

In [25]: readline("my_file.txt", keep=true)
Out[25]: "Hello World and Julia!\n"
```

Рисунок 5. Пример использования readline()

Функция readlines(). Принципиальное отличие от предыдущей функции – считывание всех строк из файла или потока ввода-вывода. В данной функции также присутствует параметр keep и выполняет те же самые операции. Сохранение строк из файла происходит как сохранение вектора строк. Я воспользовалсья Lorem Ipsum для записи файла. Текст был взят с данного сайта: https://ru.lipsum.com/

```
In [27]: open("my_file.txt", "w") do io
             write(io, "Lorem ipsum dolor sit amet, consectetur adipiscing elit.\nQuisq
         end
Out[27]: 137

In [28]: readlines("my_file.txt")
Out[28]: 3-element Array{String,1}:
          "Lorem ipsum dolor sit amet, consectetur adipiscing elit."
          "Quisque ullamcorper tellus efficitur interdum scelerisque."
          "Suspendisse potenti."

In [29]: readlines("my_file.txt", keep=true)
Out[29]: 3-element Array{String,1}:
          "Lorem ipsum dolor sit amet, consectetur adipiscing elit.\n"
          "Quisque ullamcorper tellus efficitur interdum scelerisque.\n"
          "Suspendisse potenti.\n"
```

Рисунок 6. Пример использование readlines()

Функция readdlm(). Для того, чтобы воспользоваться данной функцией я воспользовалсь утилитой DelimitedFiles для чтения и записи файлов с разделителями. Предполагается, что столбцы разделены одним или несколькими пробелами. В качестве разделителя конца строки принимается \ n.

Также при вызове функции указывается тип данных в который преобразуется передаваемая строка. Ниже я создал файл с числами разделенными точкой с запятой, и далее я считал и преобразовал к Int32 и Float32.

```
In [56]: number_x = [1; 2; 3; 4; 5];
         number_y = [6; 7; 8; 9; 10];
         open("delim_file.txt", "w") do io
             writedlm(io, [number_x number_y])
         end;
         readdlm("delim_file.txt", Int32)

Out[56]: 5x2 Array{Int32,2}:
          1   6
          2   7
          3   8
          4   9
          5  10

In [55]: readdlm("delim_file.txt", Float32)

Out[55]: 5x2 Array{Float32,2}:
          1.0   6.0
          2.0   7.0
          3.0   8.0
          4.0   9.0
          5.0  10.0
```

Рисунок 7. Пример использования readdlm()

Функция print(). Запись происходит в io (или в стандартный выходной поток stdout, если io не задан) общепринятое (недекорированное) текстовое представление. Представление, используемое print, включает минимальное форматирование и пытается избежать специфичных для Julia деталей.

```
print("Suspendisse in dolor nisl.")
print("Sed convallis mi mauris, venenatis commodo nunc ultrices vitae.")
```

```
Suspendisse in dolor nisl.Sed convallis mi mauris, venenatis commodo nunc ult
rices vitae.
```

Рисунок 8. Пример использования print()

Функция println(). Эквивалентен функции print() только после вывода переходит на следующую строку

```
println("Hello, world")
print("Julia, hello,")
print(" how are you?")
```

```
Hello, world
Julia, hello, how are you?
```

Рисунок 9. Пример использования println()

Функция show(). Записывает текстовое представление значения x в выходной поток ввода-вывода. Новые типы T должны перегружать функцию show (io:: IO, x:: T). Представление, используемое show, как правило, включает в себя специфичное для Julia форматирование и информацию о типе.

```
show("Julia: руководство для начинающих")
```

```
"Julia: руководство для начинающих"
```

Рисунок 10. Пример использования show()

Функция write(). Записывает общепринятое двоичное представление значения в данный поток ввода-вывода или файл. Возвращает количество байтов, записанных в поток.

```
io = IOBuffer();
write(io, "Nunc pharetra elementum tempus. In non velit porttitor, interdum nu
```

```
124
```

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

Парсит строку в число. Для целочисленных типов можно указать base (по умолчанию 10). Для типов с плавающей запятой строка анализируется как десятичное число с плавающей запятой. Комплексные числа разбираются из десятичных строк формы «R ± Iim» как Комплекс (R, I) запрошенного типа; «i» или «j» также могут использоваться вместо «im», также разрешены «R» или «Iim». Если строка не содержит допустимого числа, возникает ошибка.

```
println(parse(Int, "4567"))
print(parse(Complex{Float64}, "32.2e-1 + 9.8im"))
```

```
4567
3.22 + 9.8im
```

Рисунок 11. Пример использования parse()

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.

Информацию о языке Julia я нашел на данном ресурсе: http://ihed.ras.ru/~thermo/Julia/Brief%20description%20of%20Julia%20language.pdf

Объявим 2 переменные и запишем в них целые числа. Далее я сделал все перечисленные операции. К операциям сравнения можно еще добавить больше или равно (>=) и меньше или равно (<=).

```
a = 9
b = 10
print("Сложение: ")
println(a + b)
println("Унарный плюс +a = ", +a)
println("Унарный минус -b = ", -b)
print("Вычитание a - b = ")
println(a - b)
print("Умножение a * b = ")
println(a * b)
print("Деление b / a = ")
println(b / a)
print("Возведение в степень b^a = ")
println(b ^ a)
print("Извлечение корня из a = ")
println(sqrt(a))
println("Логические операторы: ")
println("равны ли a и b - ", a == b)
println("не равны a и b - ", a !== b)
println("a меньше b - ", a < b)
println("a больше b - ", a > b)
print("Битовые операторы: ")
println("Битовое отрицание a = ", ~a)
println("Битовое and = ", a & b)
println("Битовое or = ", a | b)
println("Оператор побитового сдвига вправо = ", a >> b)
println("Оператор побитового сдвига без знака = ", a >>> b)
println("Оператор побитового сдвига влево = ", a << b)
```

Рисунок 12. Примеры арифметических операций с данными

Вывод:

```
Сложение: 19
Унарный плюс +a = 9
Унарный минус -b = -10
Вычитание a - b = -1
Умножение a * b = 90
Деление b / a = 1.1111111111111112
Возведение в степень b^a = 1000000000
Извлечение корня из a = 3.0
Логические операторы:
равны ли a и b - false
не равны a и b - true
a меньше b - true
a больше b - false
Битовые операторы: Битовое отрицание a = -10
Битовое and = 8
Битовое or = 11
Оператор побитового сдвига вправо = 0
Оператор побитового сдвига без знака = 0
Оператор побитового сдвига влево = 9216
```

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

Создадим 2 матрицы одинакового размера 3x3 для удобства.

```
In [45]: # Создать матрицу L
         L = [1 2 3;4 5 6;7 8 9]

Out[45]: 3×3 Array{Int64,2}:
          1  2  3
          4  5  6
          7  8  9

In [46]: R = [5 5 2; 1 2 3; 2 3 4]

Out[46]: 3×3 Array{Int64,2}:
          5  5  2
          1  2  3
          2  3  4
```

Далее идут привычные нам операции – сложение (+), вычитание (-), умножение (*), и транспонирование (‘).

```
In [47]: # сложение 2 матриц
         L + R

Out[47]: 3×3 Array{Int64,2}:
          6   7   5
          5   7   9
          9  11  13

In [48]: # вычитание 2 матриц
         R - L

Out[48]: 3×3 Array{Int64,2}:
           4   3  -1
          -3  -3  -3
          -5  -5  -5

In [49]: # умножение 2 матриц
         R * L

Out[49]: 3×3 Array{Int64,2}:
          39  51  63
          30  36  42
          42  51  60

In [50]: # транспонирование матрицы R
         R'

Out[50]: 3×3 LinearAlgebra.Adjoint{Int64,Array{Int64,2}}:
          5  1  2
          5  2  3
          2  3  4
```

Рисунок 13. Основные операции с матицами

Теперь создадим скаляр. Допустим, что он равен 8.

```
# введем скаляр Q
Q = 3
```

```
3
```

Теперь умножим матрицу R на данный скаляр Q.

```
# умножение матрицы R на скаляр Q
R * Q
```

```
3×3 Array{Int64,2}:
 15  15   6
  3   6   9
  6   9  12
```

Выводы

Подготовил рабочее пространство и инструментарии для работы с языком программирования Julia, на простейших примерах познакомился с основами синтаксиса Julia.

3. Вывод