Лабораторная работа #1. Julia. Установка и настройка. Основные принципы.

Хохлачева Яна, учебная группа: НКНбд-01-18

Содержание

Список иллюстраций

Цель работы

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

Выполнение работы

Синтаксис языка Julia на примерах

Используем функцию typeof() для определения типа числовой величины.

В Julia введены специальные значения Inf, -Inf, NaN, обозначающие бесконечность и отсутствие какого-либо значения. Такие значения могут получаться в результате операций типа деления на ноль, а также могут быть допустимой частью выражений, поскольку в языке имеют тип вещественного числа.

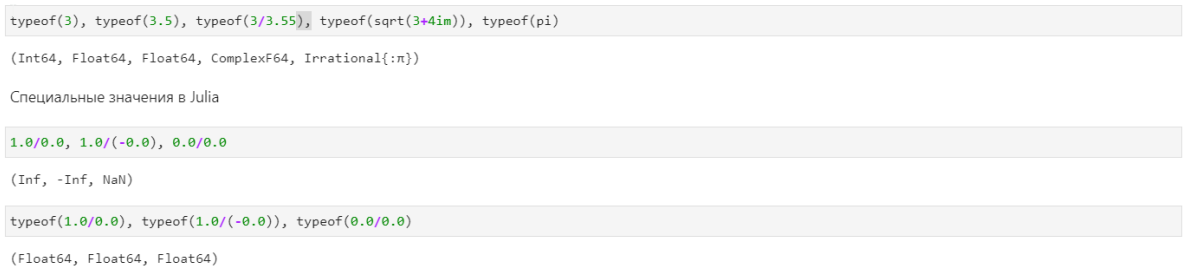

Рис. 0.1: Использование функции typeof() и специальные величины в Julia

Код для определения крайних значений диапазонов целочисленных числовых величин

Ниже представлен код для определения крайних значений диапазонов целочисленных числовых величин

```
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. 0.2: Определение крайних диапазонов целочисленных величин

Преобразование типов реализуется или прямым указанием, или с использованием обобщённого оператора convert()

В Julia преобразование типов можно реализовать или прямым указанием, или с использованием обобщённого оператора преобразования типов convert()

```
Int64(2.0), Char(2), typeof(Char(2))
```

```
(2, '\x02', Char)
```

```
convert(Int64, 2.0), convert(Char,2)
```

```
(2, '\x02')
```

Для приведения нескольких аргументов к одному типу, если это возможно, используется оператор promote()

```
typeof(promote(Int8(1), Float16(4.5), Float32(4.1)))
```

```
Tuple{Float32, Float32, Float32}
```

Рис. 0.3: Преобразование типов

В Julia есть несколько методов опредения функций

Методы опредения функций

```
function f(x)
    x^2
end
```

```
f (generic function with 1 method)
```

```
f(5)
```

```
25
```

```
g(x)=x^2
```

```
g (generic function with 1 method)
```

Рис. 0.4: Методы определения функций

Синтаксис определения одномерных и двумерных массивов в Julia и обращение к их элементам

Пример определения одномерных массивов (вектор-строка и вектор-столбец) и обращение к их вторым элементам

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

```
(7, 2)
```

Пример определения двумерного массива (матрицы) и обращение к его элементам

```
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
```

```
(1, 2, 3, 4)
```

Пример выполнения операций над массивами (aa' — транспонирование вектора)

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

```
1×1 Matrix{Int64}:
 27
```

```
aa, AA, aa'
```

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

Рис. 0.5: Примеры работы с массивами

Примеры и описание основных функций Julia

Функция read() выполняет чтение из буфера памяти и чтение из файла

```
io = IOBuffer("Words")
text = read(io, String)
```

```
"Words"
```

```
fileread = read("text.txt", String)
```

```
"word1\nword2\n3\n4"
```

```
println(fileread)
```

```
word1
word2
3
4
```

Рис. 0.6: Пример применения функции read()

Функция readline() выполняет чтение из командной строки, а также первой строки из файла, функция readlines() выполняет чтение из файла

Функция readline() - чтение из командной строки, а также первой строки из файла

```
println("Как вас зовут?")
name = readline()
print("Привет, $name")
```

```
Как вас зовут?
stdin>  student
Привет, student
```

```
readline("text.txt")
```

```
"word1"
```

Функция readlines() - чтение из файла

```
lines = readlines("text.txt")
println(lines)
```

```
["word1", "word2", "3", "4"]
```

Рис. 0.7: Пример применения функций readline() и readlines()

Функция show() выводит строку в кавычках, также можно использовать для определения символа по номеру, функция write() используется для вывода и указывает количество символов

Функция show() - выводит строку в кавычках, также можно вывести символ по номеру

```
show_example = show("String")
```

```
"String"
```

```
example = "String"
show(example[2])
```

```
't'
```

Функция write() - вывод и указание числа символов, а также запись в файл

```
write(stdout, "Something")
```

```
Something
9
```

```
write("text1.txt", "Students")
println(readlines("text1.txt"))
```

```
["Students"]
```

Рис. 0.8: Пример применения функций show() и write()

Функция parse() используется для изменения типа данных. Может быть использована для перевода вводимых данных, так как ввод всегда осуществляется в формате String.

Функция parse() и пример использования

```
liniya = "312"
typeof(liniya)
```

```
String
```

```
println("Введите новое число")
d = Meta.parse(readline())
println("Число - ", d)
println("Тип - ", typeof(d))
```

```
Введите новое число
stdin>  12
Число - 12
Тип - Int64
```

Рис. 0.9: Пример применения функции parse()

Синтаксис Julia для базовых математических операций

- Примеры операций сложения, вычитания, умножения, деления, возведения в степень, нахождения остатка от числа, целочисленного деления, получения корня от числа.

```julia
5 + 3
```

```
8
```

```julia
5 - 4
```

```
1
```

```julia
5 * 10
```

```
50
```

```julia
10 / 5
```

```
2.0
```

```julia
10 ^ 2
```

```
100
```

```julia
21 % 2
```

```
1
```

```julia
11 ÷ 2 # \div + tab
```

```
5
```

```julia
sqrt(4) # или \sqrt + tab
```

```
2.0
```

Рис. 0.10: Базовые математические операции в Julia

В Julia представлены следующие операторы сравнения:

```
false && true # AND
```

```
false
```

```
false || true # OR
```

```
true
```

```
1.0 == 1 # Равенство значений
```

```
true
```

```
1.0 === 1 # Сравенение программного представления
```

```
false
```

Рис. 0.11: Операторы сравнения в Julia

Операции над матрицами

Матрицы и вектора в Julia заполняются следующими методами:

Заполнение матрицы по строкам и столбцам

```
AA = [1 2 3; 4 5 6; 7 8 9]
```

```
3×3 Matrix{Int64}:
 1  2  3
 4  5  6
 7  8  9
```

```
BB = [[1, 2, 3] [4, 5, 6] [7, 8, 9]]
```

```
3×3 Matrix{Int64}:
 1  4  7
 2  5  8
 3  6  9
```

Вектор-строка и вектор-столбец

```
a1 = [1 2 3]
```

```
1×3 Matrix{Int64}:
 1  2  3
```

```
a2 = [1, 2, 3]
```

```
3-element Vector{Int64}:
 1
 2
 3
```

Рис. 0.12: Объявление массивов и векторов

Транспонирование матрицы, сложение и вычитание матриц:

Транспонирование матрицы

```
CC = AA'
```

```
3×3 adjoint(::Matrix{Int64}) with eltype Int64:
 1  4  7
 2  5  8
 3  6  9
```

Сложение и вычитание матриц

```
AA + BB
```

```
3×3 Matrix{Int64}:
  2   6  10
  6  10  14
 10  14  18
```

```
AA - BB
```

```
3×3 Matrix{Int64}:
 0  -2  -4
 2   0  -2
 4   2   0
```

Рис. 0.13: Транспонирование матрицы, сложение и вычитание

Умножение матриц, и операции с векторами

Перемножение матриц, умножение матрицы на вектор и число

```
AA * BB
```

```
3×3 Matrix{Int64}:
 14   32   50
 32   77  122
 50  122  194
```

```
AA * a2
```

```
3-element Vector{Int64}:
 14
 32
 50
```

```
AA * 2
```

```
3×3 Matrix{Int64}:
  2   4   6
  8  10  12
 14  16  18
```

Умножение вектора на число и скалярное произведение векторов

```
a1 * 3
```

```
1×3 Matrix{Int64}:
 3  6  9
```

```
a1 * a2
```

```
1-element Vector{Int64}:
 14
```

Рис. 0.14: Операции с векторами и матрицами

Выводы

- Во время выполнения лабораторной работы я подготовила инструментарий для работы и ознакомилась с языком Julia для дальнейшей работы.