Отчёт по лабораторной работе №1

Julia. Установка и настройка. Основные принципы.

Ким Реачна

Содержание

Список иллюстраций

1 Цель работы

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

2 Выполнение лабораторной работы

1. Выполнила установку Far Manager, Notepad++, Julia и Anaconda Distribution через менеджер пакетов Chocolatey. А также ознакомилась с пунктом «Основы работы в блокноте Jupyter».

2. Повторила задания из раздела 1.3.3 по основам синтаксиса Julia: определяла числовой тип, определение крайних значений диапазонов целочисленных числовых значений, попробила преобразование типов, базовый синтаксис определения функций, выполняла операции с массивами.

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

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

[5]: 1.0/0.0, 1.0/(-0.0), (0.0/0.0)

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

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

[6]: (Float64, Float64, Float64)

[7]: 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. 2.1: Примеры определения типа числовых величин

```
[8]: # преобразование прямым указанием
     Int64(2.0), Char(2), typeof(Char(2))

[8]: (2, '\x02', Char)

[9]: # преобразование обобщенным оператором
     convert(Int64, 2.0), convert(Char, 2)

[9]: (2, '\x02')

[10]: # преобразование нескольких аргументов к одному типу
      typeof(promote(Int8(1), Float16(4.5), Float32(4.1)))

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

Рис. 2.2: Примеры приведения аргументов к одному типу

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

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

[12]: f(4)

[12]: 16

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

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

[14]: g(8)

[14]: 64
```

Рис. 2.3: Примеры определения функций

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

[15]: (7, 2)

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

[16]: 2×2 Matrix{Int64}:
       1  2
       3  4

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

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

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

[18]: 1×1 Matrix{Int64}:
       27

[19]: aa, AA, aa'

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

Рис. 2.4: Примеры работы с массивами

2.1 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.

- read(): функция read() используется для чтения данных из стандартного ввода (клавиатуры) или из файла.

```
[27]: message = IOBuffer("Hello, world!");
      read(message, String)

[27]: "Hello, world!"
```

Рис. 2.5: Пример использования read()

- 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.

Как мы видим в начале открыла текстовый файл на запись и поместила в него строку “Hello this is lab1!” и в конце указала `\n`. Функция write выводит нам на количество байтов, записанных в поток.. Далее с помощью функции readline() я попробовала 2 способа вывода записи, без параметра keep и с ним.

```
[28]: open("myfile.txt", "w") do io
          write(io, "Hello this is lab1!\n");
      end

[28]: 20

[29]: readline("myfile.txt")

[29]: "Hello this is lab1!"

[30]: readline("myfile.txt", keep = true)

[30]: "Hello this is lab1!\n"
```

Рис. 2.6: Пример использования readline()

- readlines(): 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. Создала и записала какой-то текст в файл `my_file.txt`:

```
[1]: write("my_file.txt", "Using function write()ia!.\n Lorem Ipsum is simply dummy text of the printing and typesetting industry.\n Lorem Ipsum has been the industry's standard dummy text ever since the 1500s.")

[1]: 181

[31]: readlines("my_file.txt")

[31]: 3-element Vector{String}:
       "Using function write()ia!"
       "Lorem Ipsum is simply dummy text of the printing and typesetting industry."
       "Lorem Ipsum has been the industry's standard dummy text ever since the 1500s."

[32]: readlines("my_file.txt", keep = true)

[32]: 3-element Vector{String}:
       "Using function write()ia!\n"
       "Lorem Ipsum is simply dummy text of the printing and typesetting industry.\n"
       "Lorem Ipsum has been the industry's standard dummy text ever since the 1500s."
```

Рис. 2.7: Пример использования readlines()

- readdlm(): используется для чтения данных из текстового файла, разделенных определенным разделителем (по умолчанию - запятой) и возвращения их в виде двумерного массива. Это полезно для обработки структурированных данных, таких как таблицы.

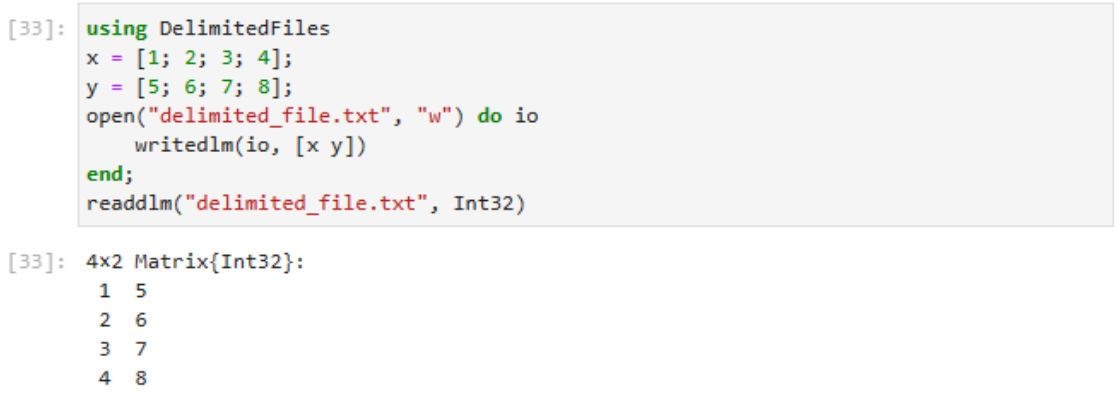

Рис. 2.8: Пример использованияя readdlm()

- print(): функция выводит аргументы на стандартный вывод без добавления символа новой строки в конце.
- println(); функция аналогична print(), но добавляет символ новой строки в конце.
- show(): функция предназначена для вывода объектов на экран. Она вызывается автоматически при использовании функций println() и print() для пользовательских типов данных.
- write(): записывает общепринятое двоичное представление значения в данный поток ввода-вывода или файл. Возвращает количество байтов, записанных в поток.

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

      Hello,World!

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

      Hello,
      World!

[36]: show("This is lab1")

      "This is lab1"

[37]: data_to_write = "Using function write()"
      write("myfile.txt", data_to_write)

[37]: 22
```

Рис. 2.9: Пример использования print(), println(), show() и write()

2. Изучите документацию по функции parse(). Функция parse() в Julia используется для преобразования строк в значения определенного типа данных.

```
[39]: str_num = "18"
      parse_int = parse(Int, str_num)
      println("Parsed integer: ", parse_int)

      Parsed integer: 18

[40]: parse(Complex{Float64}, "1.2e-1 + 3.4im")

[40]: 0.12 + 3.4im
```

Рис. 2.10: 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.

Объявите 2 переменные a = 3 и b = 4, затем вычислите арифметические операторы (+, -, *, /, ^):

[41]:
```
a = 3
b = 4
println("Сложение: ", a + b)
println("Вычитание: ", a - b)
println("Умножение: ", a * b)
println("Деление: ", a / b)
println("Возведение в степень: ", a ^ b)
println("Извлечение корня: ", sqrt(a))
```

```
Сложение: 7
Вычитание: -1
Умножение: 12
Деление: 0.75
Возведение в степень: 81
Извлечение корня: 1.7320508075688772
```

Рис. 2.11: Арифметических операций с данными

Объявите 2 переменные x = 5 и = 6, затем выполните сравнение:

[42]:
```
#  Сравнение
x = 5
y = 6
println("Is x equal to y? ", x == y)
println("Is x not equal to y? ", x != y)
println("Is x smaller than y? ", x < y)
println("Is x greater than y? ", x > y)
```

```
Is x equal to y? false
Is x not equal to y? true
Is x smaller than y? true
Is x greater than y? false
```

Рис. 2.12: Пример сравнение

Логические операции:

```
[43]: # Логические операции
      false && false

[43]: false

[44]: false && true

[44]: false

[45]: true || false

[45]: true

[46]: println("Logical AND: ", x < 10 && y < 10)
      println("Logical OR: ", x < 10 || x > 10)

      Logical AND: true
      Logical OR: true
```

Рис. 2.13: Логические операции

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

Создала 2 матрицы одинакового размера 3x3 для удобства. Далее идут привычные нам операции – сложение (+), вычитание (-), скалярное произведение, и транспонирование (’).

```
[47]: A = [1 2 3; 4 5 6; 7 8 9]

[47]: 3×3 Matrix{Int64}:
       1  2  3
       4  5  6
       7  8  9

[48]: B = [1 2 2; 2 3 3; 5 4 4]

[48]: 3×3 Matrix{Int64}:
       1  2  2
       2  3  3
       5  4  4

[49]: # сложение
      A + B

[49]: 3×3 Matrix{Int64}:
        2   4   5
        6   8   9
       12  12  13

[50]: #  вычитание
      A - B

[50]: 3×3 Matrix{Int64}:
       0  0  1
       2  2  3
       2  4  5

[51]: # скалярное произведение векторов
      v1 = [1, 2, 3]
      v2 = [4, 5, 6]
      print("Скалярное произведение векторов: ", sum(v1 .* v2))

      Скалярное произведение векторов: 32

[52]: # транспонирование матрица A
      A'

[52]: 3×3 adjoint(::Matrix{Int64}) with eltype Int64:
       1  4  7
       2  5  8
       3  6  9

[53]: # умножение на скаляр
      s = 3
      print("Матрица A, умножение на скаляр: ", A * s)

      Матрица A, умножение на скаляр: [3 6 9; 12 15 18; 21 24 27]
```

Рис. 2.14: Основные операции с матрицами и векторами

3 Листинги программы

```
#task1
# read()
message = IOBuffer("Hello, world!");
read(message, String)
"Hello, world!"
# readline()
open("myfile.txt", "w") do io
    write(io, "Hello this is lab1!\n");
end
20
readline("myfile.txt")
"Hello this is lab1!"
readline("myfile.txt", keep = true)
"Hello this is lab1!\n"
#readlines()
write("my_file.txt", "Using function write()ia!.\n Lorem Ipsum is simply dummy
    text of the printing and typesetting industry.\n Lorem Ipsum has been
    the industry's standard dummy text ever since the 1500s.")
190
readlines("my_file.txt")
3-element Vector{String}:
 "Using function write()ia!"
```

```
 "Lorem Ipsum is simply dummy text of the printing and typesetting industry."
 "Lorem Ipsum has been the industry's standard dummy text ever since the 1500s."

readlines("my_file.txt", keep = true)
3-element Vector{String}:
 "Using function write()ia!\n"
 "Lorem Ipsum is simply dummy text of the printing and typesetting industry.\n"
 "Lorem Ipsum has been the industry's standard dummy text ever since the 1500s."
#readdlm()
using DelimitedFiles
x = [1; 2; 3; 4];
y = [5; 6; 7; 8];
open("delimited_file.txt", "w") do io
    writedlm(io, [x y])
end;
readdlm("delimited_file.txt", Int32)
4×2 Matrix{Int32}:
 1  5
 2  6
 3  7
 4  8
#print() println() show() write()
print("Hello,")
print("World!")
Hello,World!
println("Hello,")
println("World!")
Hello,
World!
```

```
show("This is lab1")
```

```
"This is lab1"
```

```
data_to_write = "Using function write()"
write("myfile.txt", data_to_write)
```

```
22
```

```
#task2
?parse()
str_num = "18"
parse_int = parse(Int, str_num)
println("Parsed integer: ", parse_int)
parse(Complex{Float64}, "1.2e-1 + 3.4im")
```

```
Parsed integer: 18
0.12 + 3.4im
```

```
#task3
a = 3
b = 4
println("Сложение: ", a + b)
println("Вычитание: ", a - b)
println("Умножение: ", a * b)
println("Деление: ", a / b)
println("Возведение в степень: ", a ^ b)
println("Извлечение корня: ", sqrt(a))
```

```
Сложение: 7
Вычитание: -1
```

```
Умножение: 12
Деление: 0.75
Возведение в степень: 81
Извлечение корня: 1.7320508075688772
#  Сравнение
x = 5
y = 6
println("Is x equal to y? ", x == y)
println("Is x not equal to y? ", x != y)
println("Is x smaller than y? ", x < y)
println("Is x greater than y? ", x > y)
Is x equal to y? false
Is x not equal to y? true
Is x smaller than y? true
Is x greater than y? false
# Логические операции
false && false
false && true
true || false
println("Logical AND: ", x < 10 && y < 10)
println("Logical OR: ", x < 10 || x > 10)

false
false
true
Logical AND: true
Logical OR: true
#task4
A = [1 2 3; 4 5 6; 7 8 9]
```

```
3×3 Matrix{Int64}:
 1  2  3
 4  5  6
 7  8  9
B = [1 2 2; 2 3 3; 5 4 4]
3×3 Matrix{Int64}:
 1  2  2
 2  3  3
 5  4  4
# сложение
A + B
3×3 Matrix{Int64}:
  2   4   5
  6   8   9
 12  12  13
#  вычитание
A - B
3×3 Matrix{Int64}:
 0  0  1
 2  2  3
 2  4  5
# скалярное произведение векторов
v1 = [1, 2, 3]
v2 = [4, 5, 6]
print("Скалярное произведение векторов: ", sum(v1 .* v2))
Скалярное произведение векторов: 32
# транспонирование матрица A
A'
3×3 adjoint(::Matrix{Int64}) with eltype Int64:
```

```
 1  4  7
 2  5  8
 3  6  9
# умножение на скаляр
s = 3
print("Матрица A, умножение на скаляр: ", A * s)
```

```
Матрица A, умножение на скаляр: [3 6 9; 12 15 18; 21 24 27]
```

4 Вывод

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