РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ № 1

02.03.02 —Компьютерный практикум по статистическому анализу данных

Выполнил

Студент группы НФИбд-01-21

Студенческий билет №: 1032198038

Гань Чжаолун

15.11.2024г.

МОСКВА

2024 г.

Цель работы

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

Работы

1.1. Установите под свою операционную систему Julia, Jupyter

Я установил дистрибутивы Notepad++, Julia и Anaconda.И я также знаком с разделом "Основы работы в Jupyter Notebook".

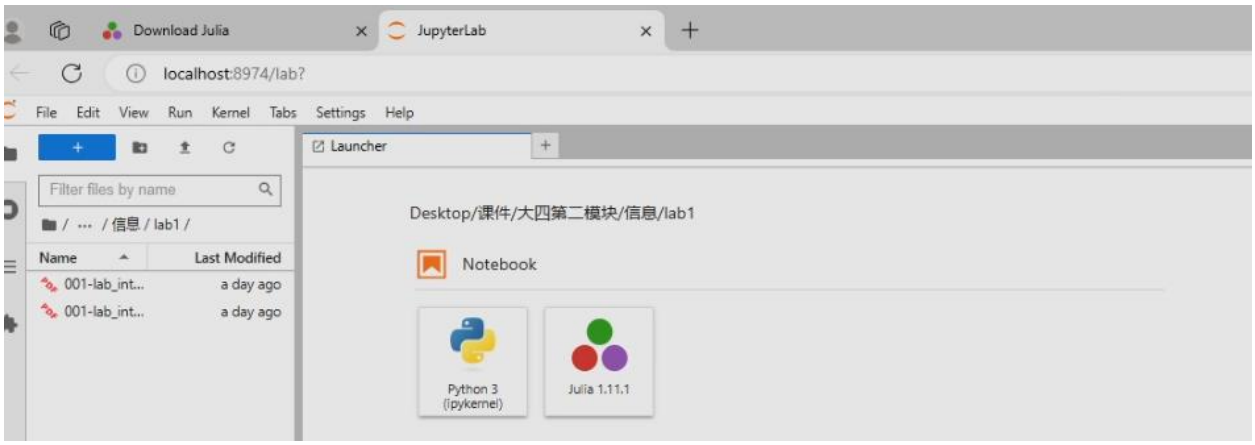

Рис. 1.1. интерфейс jupyter

1.2. Используя Jupyter Lab, повторите примеры из раздела 1.3.3.

- Функция typeof нужна для определения типа данных.

```
typeof(1), typeof(4.5), typeof(pi)
```

```
(Int64, Float64, Irrational{:π})
```

```
1.0/0.0, 1.0/(-0.0), 0.0/0.0
```

```
(Inf, -Inf, NaN)
```

```
typeof(1.0/0.0), typeof(1.0/(-0.0)), typeof(0.0/0.0)
```

```
(Float64, Float64, Float64)
```

Рис. 2.1. Пример функции typeof

- Для разных типов данных есть свои ограничения. Их можно узнать при помощи: Даны не все типы данных, в массив можно добавлять другие типы.

```
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. 2.2. Для разных типов данных есть свои ограничения

- Чтобы конвертировать из одного типа данных в другие можно использовать функцию convert или же имя типа данных, в которое надо конвертировать:

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

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

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

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

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

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

```
Bool(1), Bool(0)
```

```
(true, false)
```

Рис. 2.3. Пример функции convert

- Для приведения нескольких данных к единому типу можно использовать функцию promote:

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

```
(1.0f0, 4.5f0, 4.1f0)
```

Рис. 2.4. Пример функции promote

- Чтобы создать функцию нужно использовать ключевое слово function с указанием имени и параметров функции.

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

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

```
f(4)
```

```
16
```

Рис. 2.5. Пример функции function

- Также существует однострочный вариант записи функции:

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

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

```
g(8)
```

```
64
```

Рис. 2.6. Однострочный вариант записи функции

- В julia встроены такие типы данных, как векторы, матрицы и т.д. Для транспонирования матрицы используется символ ' после названия переменной матрицы. Матрицы создаются и используется следующим образом:

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

```
(7, 2)
```

```
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)
```

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

```
1×1 Matrix{Int64}:
 27
```

Рис. 2.7. Пример работы с массивами

1.3. Выполните 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 я могу ввести символ ‘?’ слева, чтобы просмотреть соответствующую информацию.

```
?read()
```

```
read(io::IO, T)
```
Read a single value of type `T` from `io`, in canonical binary representation.

Note that Julia does not convert the endianness for you. Use `ntoh` or `ltoh` for this purpose.

```
read(io::IO, String)
```
Read the entirety of `io`, as a `String` (see also `readchomp`).

Рис. 3.0. Пример справочная информация

• read() - считывает один символ или байт из потока ввода.

```
io = IOBuffer("Hi,bro!")
```

```
IOBuffer(data=UInt8[...], readable=true, writable=false, seekable=true, append=false, size=7, maxsize=Inf, ptr=1, mark=-1)
```

```
io = IOBuffer("Hi,bro!");
read(io, String)
```

```
"Hi,bro!"
```

Рис. 3.1. Пример функция read()

Функция read() используется для чтения данных из файла или потока. Её можно использовать для чтения данных в бинарном или текстовом формате. Эта функция возвращает все содержимое файла или потока.

• readline() - считывает одну строку из потока ввода.

```
write("GZL_lab1.txt", "你好.\n我是甘兆隆.\n");
readline("GZL_lab1.txt")
```

```
"你好."
```

```
write("GZL_lab1.txt", "你好.\n我是甘兆隆.\n");
readline("GZL_lab1.txt",keep=true)
```

```
"你好.\n"
```

Рис. 3.2. Пример функция readline()

Функция readline() используется для чтения одной строки из файла или 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.

• readlines() - считывает все строки из потока ввода и возвращает массив строк.

```
readlines("GZL_lab1.txt")
```

```
2-element Vector{String}:
 "你好."
 "我是甘兆隆."
```

Рис. 3.3. Пример функция readlines()

Функция readline() читает все строки из файла или потока и возвращает их в виде массива строк.

Каждая строка файла становится элементом массива.

• readdlm() - считывает данные из файла с разделителями и возвращает их в виде массива или матрицы.

я воспользовалась утилитой Delimited File s для чтения и записи файлов с разделителями.

```
using DelimitedFiles
x = [1; 2; 3; 4; 5];
y = [6; 7; 8; 9; 10];
open("delim_file.txt", "w") do io
    writedlm(io, [x y])
end;
readdlm("delim_file.txt", Int32)
```

```
5×2 Matrix{Int32}:
 1   6
 2   7
 3   8
 4   9
 5  10
```

```
readdlm("delim_file.txt", Float32)
```

```
5×2 Matrix{Float32}:
 1.0   6.0
 2.0   7.0
 3.0   8.0
 4.0   9.0
 5.0  10.0
```

Рис. 3.4. Пример функция readdlm()

Функция readln() используется для чтения данных, разделённых определёнными разделителями (например, запятыми, табуляцией и т.д.).

• print() - выводит значения, разделяя их пробелами.

```
io = IOBuffer();
print(io, "Hi", ' ', :bro!)
String(take!(io))
```

```
"Hi bro!"
```

```
print("Hi, bro!")
```

```
Hi, bro!
```

Рис. 3.5. Пример функция print()

Функция print() выводит переданное значение в стандартный вывод. Она не добавляет символ новой строки после вывода.

• println() - выводит значения, добавляя перевод строки в конце.

```
println("Hi, bro")
```

```
Hi, bro
```

```
io = IOBuffer();
println(io, "Hi", ',', " bro.")
String(take!(io))
```

```
"Hi, bro.\n"
```

Рис. 3.6. Пример функция println()

Функция println() аналогична print(), но после вывода добавляет символ новой строки.

• show() - отображает структуру данных с полной информацией о их содержимом.

```
show("Hi, bro!")
```

```
"Hi, bro!"
```

```
struct 月
           n::Int
       end
Base.show(io::IO, ::MIME"text/plain", d::月) = print(io, d.n, "月")
月(4)
```

```
4月
```

Рис. 3.7. Пример функция show()

Функция show() используется для вывода представления значения, которое может быть полезным для разработчиков. Она может выводить данные в формате, более подходящем для отладки.

• write() - записывает данные в поток вывода.

```
io = IOBuffer();
write(io, "Hi.", " bor.")
String(take!(io))
```

```
"Hi. bor."
```

```
write(io, "Hi.")*write(io, " bor.")
String(take!(io))
```

```
"Hi. bor."
```

Рис. 3.8. Пример функция write()

Функция write() используется для записи данных в файл или поток. Она записывает данные в бинарной форме, но также может записывать строки.

Отличия между функциями:

Чтение данных:

- read() — читает всё содержимое целиком.
- readline() — читает одну строку.
- readlines() — читает все строки и возвращает массив строк.
- readdlm() — читает данные, разделённые определёнными символами.

Запись и вывод данных:

- print() — выводит данные без перехода на новую строку.
- println() — выводит данные с переходом на новую строку.
- show() — выводит данные в формате, удобном для отладки.
- write() — записывает данные в файл или поток (бинарный или текстовый формат).

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

Функция parse() в языке программирования Julia используется для преобразования строки в соответствующий ей тип данных. Например, она может преобразовать строку в целое число, число с плавающей запятой или другой тип данных.

```
parse(Int, "4568")
```

```
4568
```

```
parse(Int, "4568", base=16)
```

```
17768
```

```
parse(Int, "10111111010000101111001", base=2)
```

```
3133625
```

Рис. 3.9. Пример функция 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.

В Julia арифметические операции могут быть применены как к целым числам, так и к числам с плавающей запятой, с результатом соответствующего типа. Оператор деления / всегда возвращает число с плавающей точкой. Для возведения в степень используется символ ", а для извлечения квадратного корня - функция sqrt(). Логические операции выполняются с помощью операторов && (логическое "и") и || (логическое "или").также можно добавить к операции сравнения значение, большее или равное (>=), и значение, меньшее или равное (<=).

```
a=5
b=3

# сложение
println(a+b)

# вычитание
println(a-b)

# умножение
println(a*b)

# деление
println(a/b)

# деление с округлением до целого числа
println(div(a,b))
```

```
8
2
15
1.6666666666666667
1
```

```
# возведение в степень
println(a^b)

# извлечение корня
println(sqrt(25))
```

```
125
5.0
```

```
# сравнение

x=10
y=20

# равенство
println(x==y)

# неравенство
println(x!=y)

# больше или меньше
println(x<y)
println(x>y)

# больше или равно, меньше или равно
println(x<=y)
println(x>=y)
```

```
false
true
true
false
true
false
```

```
# логические операции

# and
println((x<y)&&(y>15))

# or
println((x>y)||(y==20))

# no
println(!(x==y))
```

```
true
true
true
```

Рис. 3.10. Примеры арифметических операций с данными

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

Действия с матрицами, векторами и т.д.

- Сложение и вычитание матриц выполняются покомпонентно.
- Скалярное произведение векторов вычисляется как сумма произведений соответствующих элементов векторов.
- Транспонирование матрицы приводит к замене строк на столбцы (и наоборот).
- Умножение матрицы на скаляр заключается в умножении каждого элемента матрицы на данный скаляр.

```
v1=[1, 4, 8]
v2=[6, 7, 9]

# сложение векторов
v_sum=v1+v2
println(v_sum)

# вычитание векторов
v_diff=v1-v2
println(v_diff)

# умножение на скаляр вектора
scalar=2
v_sca=scalar*v1
println(v_sca)

# скалярное произведение вектора
using LinearAlgebra

dot_product=dot(v1, v2)
println(dot_product)
```

```
[7, 11, 17]
[-5, -3, -1]
[2, 8, 16]
106
```

```
A=[1 4; 5 8]
B=[3 6; 7 9]

# сложение матрица
A_plus_B=A+B
println(A_plus_B)

# вычитание матрица
A_minus_B=A-B
println(A_minus_B)

# умножение на матрицы скаляр
scalar=3
A_sca=scalar*A
println(A_sca)
```

```
[4 10; 12 17]
[-2 -2; -2 -1]
[3 12; 15 24]
```

```
v=[3, 9]
A=[1 4; 5 8]
B=[2 0; 1 3]

# умножение матрицы на вектор
A_v_product=A*v
println(A_v_product)

# умножение 2 матриц
A_B_product=A*B
println(A_B_product)
```

```
[39, 87]
[6 12; 18 24]
```

```
# транспонирование
v=[4, 5, 8]
A=[1 4; 5 8]

# транспонирование вектора
v_transposed=v'
println(v_transposed)

# транспонирование матрицы
A_transposed=A'
println(A_transposed)
```

```
[4 5 8]
[1 5; 4 8]
```

Рис. 3.11. Примеры основные операции с матицами

Вывод

По итогам выполнения работы, я выполнил задание лабораторной работы. Я подготовил рабочую область и инструменты для использования языка программирования Julia и использовал простейшие примеры, чтобы ознакомиться с основами грамматики Julia.