

# Лабораторная работа №2. Компьютерный практикум по статистическому анализу данных

Леснухин Даниил Дмитриевич  
Российский университет дружбы народов  
Москва

## Цель работы

Основная цель работы – изучить несколько структур данных, реализованных в Julia, научиться применять их и операции над ними для решения задач.

## Задание

1. Используя Jupyter Lab, повторите примеры.
2. Выполните задания для самостоятельной работы.

## Теоретическое введение

Julia – высокоуровневый свободный язык программирования с динамической типизацией, созданный для математических вычислений [[@julialang](#)]. Эффективен также и для написания программ общего назначения. Синтаксис языка схож с синтаксисом других математических языков, однако имеет некоторые существенные отличия.

Для выполнения заданий была использована официальная документация Julia [[@juliadoc](#)].

Рассмотрим несколько структур данных, реализованных в Julia. Несколько функций (методов), общих для всех структур данных:

- `isempty()` – проверяет, пуста ли структура данных;
- `length()` – возвращает длину структуры данных;
- `in()` – проверяет принадлежность элемента к структуре;
- `unique()` – возвращает коллекцию уникальных элементов структуры,
- `reduce()` – свёртывает структуру данных в соответствии с заданным бинарным оператором;
- `maximum()` (или `minimum()`) – возвращает наибольший (или наименьший) результат вызова функции для каждого элемента структуры данных.

# Выполнение лабораторной работы

Для начала выполним примеры из раздела про кортежи

СЛОВАРИ

[37]  
✓  
0  
СМК.

# создать словарь с именем phonebook:  
phonebook = Dict("Иванов И.И." => ("867-5309", "333-5544"), "Бухгалтерия" => ("555-2368", "867-5309"))

▼

Dict{String, Any} with 2 entries:  
"Бухгалтерия" => "555-2368"  
"Иванов И.И." => ("867-5309", "333-5544")

[38]  
✓  
0  
СМК.

# вывести значения элементов словаря:  
values(phonebook)

▼

ValueIterator for a Dict{String, Any} with 2 entries. Values:  
"555-2368"  
("867-5309", "333-5544")

[39]  
✓  
0  
СМК.

# вывести значения элементов словаря:  
values(phonebook)

▼

ValueIterator for a Dict{String, Any} with 2 entries. Values:  
"555-2368"  
("867-5309", "333-5544")

[40]  
✓  
0  
СМК.

# вывести заданные в словаре пары "ключ - значение":  
pairs(phonebook)

▼

Dict{String, Any} with 2 entries:  
"Бухгалтерия" => "555-2368"  
"Иванов И.И." => ("867-5309", "333-5544")

[41]  
✓  
0  
СМК.

# проверка вхождения ключа в словарь:  
haskey(phonebook, "Иванов И.И.")  
# добавить элемент в словарь:  
phonebook["Сидоров П.С."] = "555-3344"

▼

"555-3344"

[42]  
✓  
0  
СМК.

# удалить ключ и связанные с ним значения из словаря  
pop!(phonebook, "Иванов И.И.")

▼

("867-5309", "333-5544")

[43]  
✓  
0  
СМК.

# Объединение словарей (функция merge()):  
a = Dict{"foo" => 0.0, "bar" => 42.0};  
b = Dict{"baz" => 17, "bar" => 13.0};  
merge(a, b), merge(b, a)

▼

(Dict{String, Real}{"bar" => 13.0, "baz" => 17, "foo" => 0.0}, Dict{String, Any}{"bar" => 42.0, "baz" => 17})

Теперь выполним примеры из раздела про словари

```
# пустой кортеж:
()

... ()

# кортеж из элементов типа String:
favoritelang = ("Python", "Julia", "R")

("Python", "Julia", "R")

# кортеж из целых чисел:
x1 = (1, 2, 3)

(1, 2, 3)

# кортеж из элементов разных типов:
x2 = (1, 2.0, "tmp")

... (1, 2.0, "tmp")

# именованный кортеж:
x3 = (a=2, b=1+2)

(a = 2, b = 3)

# длина кортежа x2:
length(x2)

3

# обратиться к элементам кортежа x2:
x2[1], x2[2], x2[3]

(1, 2.0, "tmp")

# произвести какую-либо операцию (сложение)
# с вторым и третьим элементами кортежа x1:
c = x1[2] + x1[3]

5

# обращение к элементам именованного кортежа x3:
x3.a, x3.b, x3[2]
```

Рис. 1: Примеры из раздела кортежи

```
МНОЖЕСТВА

[0] # создать множество из четырех целочисленных значений:
✓ 0 сек. A = Set([1, 3, 4, 5])

Set[Int64] with 4 elements:
5
4
3
1

[0] # создать множество из 11 символьных значений:
✓ 0 сек. B = Set("abracadabra")

Set[Char] with 5 elements:
'a'
'd'
'r'
'k'
'b'

[0] # проверка эквивалентности двух множеств:
✓ 0 сек. S1 = Set([1,2]);
S2 = Set([3,4]);
isetequal(S1,S2)
S3 = Set([1,2,2,3,1,2,3,2,1]);
S4 = Set([2,3,1]);
isetequal(S3,S4)

true

[0] # объединение множеств:
✓ 0 сек. C=union(S1,S2)

Set[Int64] with 4 elements:
4
2
3
1

[0] # пересечение множеств:
✓ 0 сек. D = intersect(S1,S3)

Set[Int64] with 2 elements:
2
1

[0] # разность множеств:
✓ 0 сек. E = setdiff(S3,S1)

Set[Int64] with 1 element:
3

[0] # проверка вхождения элементов одного множества в другое:
✓ 0 сек. issubset(S1,S4)

true

[0] # добавление элемента в множество:
✓ 0 сек. push!(S4, 99)

Set[Int64] with 4 elements:
2
99
3
1

[0] # удаление последнего элемента множества:
✓ 0 сек. pop!(S4)

2
```

Выполним примеры из раздела про множества

Выполним примеры из раздела про массивы

Теперь перейдем к выполнению заданий.

### Задание №1

Даны множества:  $A = 0, 3, 4, 9$ ,  $B = 1, 3, 4, 7$ ,  $C = 0, 1, 2, 4, 7, 8, 9$ . Найдем  $P = A \cap B \cup A \cap B \cup A \cap C \cup B \cap C$

### Задание №2

Приведем свои примеры с выполнением операций над множествами элементов разных типов

▼ МАССИВЫ

[59]  
✓ O  
сбк.

# создание пустого массива с абстрактным типом:  
empty\_array\_1 = []

▼  
Any[]

[61]  
✓ O  
сбк.

# создание пустого массива с конкретным типом:  
empty\_array\_2 = (Int64)[]  
empty\_array\_3 = (Float64)[]

▼  
Float64[]

[62]  
✓ O  
сбк.

# вектор-столбец:  
a = [1, 2, 3]

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

[63]  
✓ O  
сбк.

# вектор-строка:  
b = [1 2 3]

▼  
1x3 Matrix{Int64}:  
1 2 3

[64]  
✓ O  
сбк.

# многомерные массивы (матрицы):  
A = [[1, 2, 3] [4, 5, 6] [7, 8, 9]]  
B = [[1 2 3]; [4 5 6]; [7 8 9]]

▼  
3x3 Matrix{Int64}:  
1 2 3  
4 5 6  
7 8 9

[69]  
✓ O  
сбк.

# одномерный массив из 8 элементов (массив \$1 \times 8\$)  
# со значениями, случайно распределёнными на интервале [0, 1]:  
c = rand(1,8)

▼  
1x8 Matrix{Float64}:  
0.0859245 0.479945 0.162709 0.0130608 \_ 0.646947 0.973238 0.871803

[71]  
✓ O  
сбк.

# многомерный массив \$2 \times 3\$ (2 строки, 3 столбца) элементов  
# со значениями, случайно распределёнными на интервале [0, 1]:  
C = rand(2,3)

▼  
2x3 Matrix{Float64}:  
0.178175 0.725235 0.0246914  
0.174706 0.126942 0.565709

Рис. 2: Примеры использования массивов

```
[72] # трёхмерный массив:
0 D = rand(4, 3, 2)
СВК.

*** 4x3x2 Array{Float64, 3}:
[:, :, 1] =
 0.667413  0.327224  0.995605
 0.670233  0.866103  0.894114
 0.473628  0.707488  0.201904
 0.677968  0.270496  0.652811

[:, :, 2] =
 0.0493652  0.419159  0.194344
 0.575047  0.94959  0.493161
 0.625627  0.935972  0.0933416
 0.486154  0.958258  0.0225988

[73] # массив из квадратных корней всех целых чисел от 1 до 10:
0 roots = [sqrt(i) for i in 1:10]
СВК.

*** 10-element Vector{Float64}:
 1.0
 1.4142135623730951
 1.7320508075688772
 2.0
 2.23606797749979
 2.449489742783178
 2.6457513110645907
 2.8284271247461903
 3.0
 3.1622776601683795

[ ] # массив с элементами вида 3*x^2,
# где x - нечётное число от 1 до 9 (включительно)

[74] ar_1 = [3*i^2 for i in 1:2:9]
0
СВК.

*** 5-element Vector{Int64}:
 3
 27
 75
 147
 243

[75] # массив квадратов элементов, если квадрат не делится на 5 или 4:
0 ar_2=[i^2 for i=1:10 if (i^2%5!=0 && i^2%4!=0)]
СВК.

*** 4-element Vector{Int64}:
 1
 9
 49
 81
```

Рис. 3: Примеры использования массивов

```
76] # одномерный массив из пяти единиц:
0   ones(5)
сек.

5-element Vector{Float64}:
 1.0
 1.0
 1.0
 1.0
 1.0

77] # двумерный массив 2x3 из единиц:
    ones(2,3)

... 2x3 Matrix{Float64}:
 1.0  1.0  1.0
 1.0  1.0  1.0

78] # одномерный массив из 4 нулей:
0   zeros(4)
сек.

... 4-element Vector{Float64}:
 0.0
 0.0
 0.0
 0.0

79] # заполнить массив 3x2 цифрами 3.5
0   fill(3.5, (3,2))
сек.

3x2 Matrix{Float64}:
 3.5  3.5
 3.5  3.5
 3.5  3.5

80] # заполнение массива посредством функции repeat():
0   repeat([1,2],3,3)
сек.
    repeat([1 2],3,3)

... 3x6 Matrix{Int64}:
 1  2  1  2  1  2
 1  2  1  2  1  2
 1  2  1  2  1  2

81] # преобразование одномерного массива из целых чисел от 1 до 12
0   # в двумерный массив 2x6
сек.
    a = collect(1:12)
    b = reshape(a, (2,6))

... 2x6 Matrix{Int64}:
 1  3  5  7  9 11
 2  4  6  8 10 12

82] # транспонирование
0   b'
сек.
    # транспонирование
    c = transpose(b)
```

Рис. 4: Примеры использования массивов

```
# массив 10x5 целых чисел в диапазоне [10, 20]:
ar = rand(10:20, 10, 5)

*** 10x5 Matrix{Int64}:
 15 19 15 18 11
 15 16 13 19 20
 13 19 12 11 14
 17 20 14 11 12
 13 16 17 17 17
 11 17 12 13 19
 13 14 17 16 10
 17 20 19 15 10
 15 20 19 14 19
 13 18 11 16 16

# выбор всех значений строки в столбце 2:
ar[:, 2]

10-element Vector{Int64}:
 19
 16
 19
 20
 16
 17
 14
 20
 20
 18

# выбор всех значений в столбцах 2 и 5:
ar[:, [2, 5]]

10x2 Matrix{Int64}:
 19 11
 16 20
 19 14
 20 12
 16 17
 17 19
 14 10
 20 10
 20 19
 18 16

# все значения строк в столбцах 2, 3 и 4:
ar[:, 2:4]

10x3 Matrix{Int64}:
 19 15 18
 16 13 19
 19 12 11
 20 14 11
 16 17 17
 17 12 13
 14 17 16
 20 19 15
 20 19 14
 18 11 16
```

Рис. 5: Примеры использования массивов



```
[88]
✓ O
Cbk.
# значения в строках 2, 4, 6 и в столбцах 1 и 5:
ar[[2, 4, 6], [1, 5]]

3x2 Matrix{Int64}:
 15 20
 17 12
 11 19

[89]
✓ O
Cbk.
# значения в строке 1 от столбца 3 до последнего столбца:
ar[1, 3:end]

3-element Vector{Int64}:
 15
 18
 11

[90]
✓ O
Cbk.
# сортировка по столбцам:
sort(ar, dims=1)

10x5 Matrix{Int64}:
 11 14 11 11 10
 13 16 12 11 10
 13 16 12 13 11
 13 17 13 14 12
 13 18 14 15 14
 15 19 15 16 16
 15 19 17 16 17
 15 20 17 17 19
 17 20 19 18 19
 17 20 19 19 20

[91]
✓ O
Cbk.
# поэлементное сравнение с числом
# (результат - массив логических значений):
ar .> 14

10x5 BitMatrix:
 1 1 1 1 0
 1 1 0 1 1
 0 1 0 0 0
 1 1 0 0 0
 0 1 1 1 1
 0 1 0 0 1
 0 0 1 1 0
 1 1 1 1 0
 1 1 1 0 1
 0 1 0 1 1

[92]
✓ O
Cbk.
# возврат индексов элементов массива, удовлетворяющих условию:
findall(ar .> 14)

30-element Vector{CartesianIndex{2}}:
 CartesianIndex{1, 1}
 CartesianIndex{2, 1}
 CartesianIndex{4, 1}
 CartesianIndex{8, 1}
 CartesianIndex{9, 1}
 CartesianIndex{1, 2}
 CartesianIndex{2, 2}
 CartesianIndex{3, 2}
 CartesianIndex{4, 2}
 CartesianIndex{5, 2}
 CartesianIndex{6, 2}
 CartesianIndex{8, 2}
 CartesianIndex{9, 2}
 CartesianIndex{1, 3}
 CartesianIndex{2, 3}
 CartesianIndex{3, 3}
 CartesianIndex{4, 3}
 CartesianIndex{5, 3}
 CartesianIndex{6, 3}
 CartesianIndex{7, 3}
 CartesianIndex{8, 3}
 CartesianIndex{9, 3}
 CartesianIndex{1, 4}
 CartesianIndex{2, 4}
 CartesianIndex{3, 4}
 CartesianIndex{4, 4}
 CartesianIndex{5, 4}
 CartesianIndex{6, 4}
 CartesianIndex{7, 4}
 CartesianIndex{8, 4}
 CartesianIndex{9, 4}
```

Рис. 6: Примеры использования массивов

```

▶ A = Set([0, 3, 4, 9]);
  B = Set([1, 3, 4, 7]);

  C = Set([0, 1, 2, 4, 7, 8, 9]);

▶ P = union(intersect(A, B), intersect(A, B), intersect(A, C), intersect(B, C))

... Set{Int64} with 6 elements:
    0
    4
    7
    9
    3
    1

```

Рис. 7: Задание №1. Работа с множествами

### Задание №3

Создадим массивы разными способами, используя циклы

```

3

[113] 0 con. ▶ d = [i for i in 1:21]
print(d, '\n')
... [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21]

[114] 0 con. c = []
c_1 = reverse([i for i in 1:21])
print(c_1, '\n')
... [21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

[115] 0 con. print(vcat(d, c_1), '\n')
... [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

[116] 0 con. tmp = [4 6 3]
println(vcat(fill.(tmp, [10 2 1])).)
... [[4, 4, 4, 4, 4, 4, 4, 4, 4, 4] [6, 6] [3]]

[117] 0 con. println(vcat(fill.(tmp, [10 10 10])).)
... [4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3]

[118] 0 con. println(vcat(fill.(tmp, [11 10 10])).)
... [4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3]

[119] 0 con. println(vcat(fill.(tmp, [10 20 30])).)
... [4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3]

[121] 0 con. ▶ tmp_deg = []
for i in 1:3
    push!(tmp_deg, 2^tmp[i])
end

println(vcat(fill.(tmp_deg, [1, 1, 4])).)
count = 0
for i in tmp_deg
    if '6' in string(i)
        count = count + 1
    end
end
println(count)
... [16, 64, 8, 8, 8, 8]
2

```

```
2

[109] Set1 = Set([1, 2, 3, "Hello"])
✓ 0 ОЕК.
  Set{Any} with 4 elements:
    2
    "Hello"
    3
    1

[110] println("\nElements of set1:")
✓ 0 ОЕК.
  for i in Set1
    println(i)
  end

  ...
  Elements of set1:
  2
  Hello
  3
  1

[111] print(in("Hello", Set1))
✓ 0 ОЕК.
  true

[112] Set1 = push!(Set1, "World!")
✓ 1 ОЕК.
  println("\nSet after adding one element:\n")

  for i in 1:5
    push!(Set1, i)
  end
  println("\nset after adding range of elements:\n", Set1)

  ...
  Set after adding one element:

  set after adding range of elements:
  Set{Any}[5, 4, "World!", 2, "Hello", 3, 1]
```

Рис. 8: Задание №2. Примеры операций над множествами элементов разных типов

```

3.10
[122]
0
cek.
using Statistics
y(x) = exp(x) * cos(x)
Y = [y(x) for x in 3:0.1:6]
mean(Y)

... 53.11374594642971

3.11
[125]
0
cek.
vect = []
for j in 1:3:34
    i = j*2
    push!(vect, ((0.1)^i, (0.2)^j))
end
println(vect)

Any[(0.010000000000000002, 0.2), (1.0000000000000005e-8, 0.001600000000000003), (1.000000000000008e-14, 1.280000000000005e-5), (1.0000000000000011e-20, 1.0240000000000006e-5)]

[126]
0
cek.
vect_0 = [(2^i)/i for i in 1:25]
println(vect_0)

... [2.0, 2.0, 2.6666666666666665, 4.0, 6.4, 10.666666666666666, 18.285714285714285, 32.0, 56.888888888888886, 102.4, 186.1818181818182, 341.3333333333333, 630.1538461538462, 117]

vect_1 = []
for i in 1:30
    push!(vect_1, "fn$i")
end
print(vect_1)

... Any["fn1", "fn2", "fn3", "fn4", "fn5", "fn6", "fn7", "fn8", "fn9", "fn10", "fn11", "fn12", "fn13", "fn14", "fn15", "fn16", "fn17", "fn18", "fn19", "fn20", "fn21", "fn22", "fn23", "fn24", "fn25", "fn26", "fn27", "fn28", "fn29", "fn30"]

[31]
0
cek.
vect_x = []
for i in 1:250
    push!(vect_x, rand(0:999))
end
println(vect_x)
vect_y = []
for i in 1:250
    push!(vect_y, rand(0:999))
end
println(vect_y)

... Any[901, 816, 949, 163, 559, 49, 598, 272, 911, 631, 73, 870, 436, 604, 508, 524, 501, 463, 626, 359, 352, 50, 326, 827, 150, 822, 319, 147, 380, 703, 722, 643, 964, 23, 430, 675, 500, 956, 77, 994, 598, 623, 756, 341, 523, 340, 148, 143, 933, 160, 759, 346, 411, 39, 811, 34, 593, 959, 842, 467, 313, 122, 115, 683, 694, 599, 768, 803, 617, 920]

[132]
0
cek.
for i in 1:249
    j = i + 1
    print(vect_y[j] - vect_x[i], ";")
end

... -401;140;-872;831;39;574;158;69;-388;-291;75;-727;497;-444;251;-178;-90;-424;185;-325;241;909;516;-360;163;-700;-204;536;314;-104;46;160;-347;897;369;-519;-562;-193;-12;-227;

```

## Задание №4

Создадим массив `squares`, в котором будут храниться квадраты всех целых чисел от 1 до 100

```

squares = [(i)^2 for i in 1:100]
print(squares)

... [1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361, 400, 441, 484, 529, 576, 625, 676, 729, 784, 841, 900, 961, 1024, 1089, 1156, 1225, 1296,]

```

Рис. 9: Задание №4.

## Задание №5

Подключим пакет `Primes` (функции для вычисления простых чисел). Сгенерируем массив `murprimes`, в котором будут храниться первые 168 простых чисел. Определим 89-е наименьшее простое число. Получим срез массива с 89-го до 99-го элемента включительно, содержащий наименьшие простые числа.

## Задание №6

```

5
[135] 29
✓ conc.
import Pkg
Pkg.add("Primes")

...
Updating registry at ~/.julia/registries/General.toml
Resolving package versions...
Installed IntegerMathUtils - v0.1.3
Installed Primes - v0.5.7
Updating ~/.julia/environments/v1.11/Project.toml
[27ebfcd6] + Primes v0.5.7
Updating ~/.julia/environments/v1.11/Manifest.toml
[18e54dd8] + IntegerMathUtils v0.1.3
[27ebfcd6] + Primes v0.5.7
Precompiling project...
3717.5 ms ✓ IntegerMathUtils
756.6 ms ✓ Primes
2 dependencies successfully precompiled in 11 seconds. 468 already precompiled.

[136] 0
✓ conc.
using Primes
myPrimes = primes(prime(168))
println(myPrimes)
myPrimes[89]
myPrimes[89:99]

...
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199,
11-element Vector{Int64}:
461
463
467
479
487
491
499
503
509
521
523

```

Рис. 10: Задание №5. Работа с пакетом Primes

Вычислим следующие выражения

$$\sum_{i=10}^{100} (i^3 + 4i^2);$$

$$\sum_{i=1}^M \left( \frac{2^i}{i} + \frac{3^i}{i^2} \right), M = 25;$$

$$1 + \frac{2}{3} + \left( \frac{24}{35} \right) + \left( \frac{246}{357} \right) + \dots + \left( \frac{24}{35} \dots \frac{39}{39} \right).$$

```

[137] 0
✓ conc.
sum = 0
for i in 10:100
    sum += i^3 + 4*i^2
end
println(sum)

... 26852735

[138] 0
✓ conc.
M = 25
sum = 0
for i in 1:M
    sum += 2^i/i + 3^i/i^2
end
println(sum)

... 2.1291704368143802e9

[139] 0
✓ conc.
N = 38
sum = 1
a_n = 1
for i in 2:2:N
    a_n *= i/(i+1)
    sum += a_n
end
println(sum)

... 6.976346137897618

```

## Выводы

В результате выполнения данной лабораторной работы я изучил несколько структур данных, реализованных в Julia, Научился применять их и операции над ними для решения задач.