

# Лабораторная работа № 2. Julia. Структуры данных

Компьютерный практикум по статистическому анализу данных

---

Сунгурова М.М.

23.11.24

Российский университет дружбы народов, Москва, Россия

## Информация

---

- Сунгурова Мариян Мухсиновна
- студентка группы НКНбд-01-21
- Российский университет дружбы народов

## Вводная часть

---

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

1. Используя Jupyter Lab, повторите примеры из раздела 2.2.
2. Выполните задания для самостоятельной работы (раздел 2.4).

- Язык программирования `Julia`

## Выполнение лабораторной работы

---



- Julia — высокоуровневый свободный язык программирования с динамической типизацией, созданный для математических вычислений. Эффективен также и для написания программ общего назначения. Синтаксис языка схож с синтаксисом других математических языков, однако имеет некоторые существенные отличия.

## Выполнение лабораторной работы

---

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.

Выполним примеры из лабораторной работы для действий над кортежами(рис. (fig:001?) - (fig:002?) )

```
Кортежи
Примеры кортежей
1)
>>> ()
-- ()

2)
>>> korotzhik = ("Python", "Java", "C")
-- korotzhik
('Python', 'Java', 'C')

3)
>>> a1 = (1, 2, 3)
>>> a2 = (3, 4, 5, "cor")
>>> a3 = (a1, a2)
-- a3
((1, 2, 3), (3, 4, 5, 'cor'))

Операции над кортежами
1)
>>> len(korotzhik)
-- 3

2)
>>> a1[1], a1[2], a1[3]
-- ('Java', 'C', 'cor')
```

Рис. 1: Примеры. Кортежи

```
Операции над кортежами
1)
>>> len(a1)
-- 3

2)
>>> a1[1], a1[2], a1[3]
-- (2, 3, 'cor')

3)
>>> a = a1[1] + a1[2]
-- 23

4)
>>> a1 + a2 + a3
-- (1, 2, 3, 3, 4, 5, 'cor', (1, 2, 3), (3, 4, 5, 'cor'))
```

# Выполнение примеров

Также со словарями(рис. (fig:003?) - (fig:004?))

```
Словари
Примеры словарей

phonebook = Dict{"Иванов И.И." => ("867-5309", "333-5544"), "Бухгалтерин" => "555-2368"}
[100] ✓ 0.0s

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

keys(phonebook)
[101] ✓ 0.0s

KeySet for a Dict{String, Any} with 2 entries. Keys:
  "Бухгалтерин"
  "Иванов И.И."

values(phonebook)
[102] ✓ 0.0s

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

pairs(phonebook)
[103] ✓ 0.0s

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

Рис. 3: Примеры. Словари

```
haskey(phonebook, "Иванов И.И.")
[104] ✓ 0.0s

true

phonebook["Сидоров"] = "555-3344"
[105] ✓ 0.0s

"555-3344"
```

Рассмотрим также примеры опреций над множествами(рис. (fig:005?) - (fig:006?))

```
Множества

A = Set({1, 2, 4, 5})
B = Set("abcd")

S1 = Set({1, 2})
S2 = Set({3, 4})
Issetequal(S1, S2)

[100] ✓ 0.0s
... false

S3 = Set({1, 2, 2, 3, 1, 2, 3, 2, 1})
S4 = Set({2, 3, 1})
Issetequal(S3, S4)

[101] ✓ 0.0s
... true

C = union(S1, S2)

[102] ✓ 0.0s
... Set[Int64] with 4 elements:
  4
  2
  3
  1

D = Intersect(S1, S3)

[103] ✓ 0.0s
... Set[Int64] with 2 elements:
  2
  1
```

Рис. 5: Примеры. Множества

```
▷ D = Intersect(S1, S3)
[103] ✓ 0.0s
... Set[Int64] with 2 elements:
  2
  1
```

И с массивами(рис. (fig:007?) - (fig:011?))

```
Массивы

emp_arr_1 = []
emp_arr_2 = (Int64){}
emp_arr_3 = (Float64){}

[106] ✓ 00s
... Float64[]

a = [1, 2, 3]
b = [1 2 3]

[107] ✓ 00s
... 1x3 Matrix{Int64}:
 1 2 3

A = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
B = [[1 2 3]; [4 5 6]; [7 8 9]]

[108] ✓ 00s
... 3x3 Matrix{Int64}:
 1 2 3
 4 5 6
 7 8 9

c = rand(1, 8)

[109] ✓ 00s
... 1x8 Matrix{Float64}:
 0.0118475 0.144387 0.0256888 0.457047 ... 0.19639 0.998878 0.471298
```

Рис. 7: Примеры. Массивы

```
c = rand(1, 8)

[109] ✓ 00s
... 1x8 Matrix{Float64}:
 0.0118475 0.144387 0.0256888 0.457047 ... 0.19639 0.998878 0.471298
```

# Выполнение примеров

```
ar_1 = [3*i^2 for i in 1:2:9]
[121] ✓ 0.0s
... 5-element Vector{Int64}:
 3
 27
 75
147
243

ar_2 = [i^2 for i=1:10 if (i^2%5!=0 && i^2%4!=0)]
[134] ✓ 0.1s
... 4-element Vector{Int64}:
 1
 9
49
81

ones(5)
[151] ✓ 0.0s
... 5-element Vector{Float64}:
 1.0
 1.0
 1.0
 1.0
 1.0

ones(2, 3)
[154] ✓ 0.0s
... 2x3 Matrix{Float64}:
 1.0  1.0  1.0
 1.0  1.0  1.0
```

Рис. 9: Примеры. Массивы

```
fill(3.5, (3,2))
[128] ✓ 0.0s
... 3x2 Matrix{Float64}:
 3.5  3.5
 3.5  3.5
 3.5  3.5

repeat([1, 2], 3, 3)
[131] ✓ 0.0s
... 3x3 Matrix{Int64}:
 1  2  1  2  1  2
 1  2  1  2  1  2
 1  2  1  2  1  2
```



# Выполнение примеров

```
c = transpose(b)
[112] ✓ 0.0s
... 6x2 transpose(::Matrix{Int64}) with eltype Int64:
 1 2
 3 4
 5 6
 7 8
 9 10
11 12

ar = rand(10:20, 10, 5)
[113] ✓ 0.0s
... 10x5 Matrix{Int64}:
17 15 18 17 18
19 16 13 16 14
12 14 17 17 18
17 11 10 14 18
20 10 17 18 17
12 17 16 20 11
12 10 19 13 16
16 13 10 11 11
15 19 20 19 20
11 19 15 19 20

ar[:, 2]
[114] ✓ 0.0s
... 10-element Vector{Int64}:
15
16
14
14
10
17
10
13
19
```

Рис. 11: Примеры. Массивы

```
ar[:, [2, 5]]
[115] ✓ 0.0s
... 10x2 Matrix{Int64}:
15 18
16 14
14 18
14 18
10 17
```

## Выполнение примеров

```
ar -> 14
[140] ✓ 0.0s

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

findall(ar -> 14)
[141] ✓ 0.0s

*** 32-element Vector{CartesianIndex{2}}:
 CartesianIndex{1, 1}
 CartesianIndex{2, 1}
 CartesianIndex{4, 1}
 CartesianIndex{5, 1}
 CartesianIndex{8, 1}
 CartesianIndex{9, 1}
 CartesianIndex{1, 2}
 CartesianIndex{2, 2}
 CartesianIndex{6, 2}
 CartesianIndex{9, 2}
      ⋮
 CartesianIndex{9, 4}
 CartesianIndex{10, 4}
 CartesianIndex{1, 5}
 CartesianIndex{3, 5}
 CartesianIndex{4, 5}
 CartesianIndex{5, 5}
 CartesianIndex{7, 5}
 CartesianIndex{9, 5}
 CartesianIndex{10, 5}
```

# Выполнение заданий для самостоятельной работы

Выполним задания для самостоятельной работы.

1. Даны множества:  $A = \{0, 3, 4, 9\}$ ,  $B = \{1, 3, 4, 7\}$ ,  $C = \{0, 1, 2, 4, 7, 8, 9\}$ . Найдём объединение пересечений этих множеств.(рис. (fig:014?))

```
Задание 1

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

A = Set([0, 3, 4, 9])
B = Set([1, 3, 4, 7])
C = Set([0, 1, 2, 4, 7, 8, 9])

[142] ✓ 0.0s
... Set{Int64} with 7 elements:
0
4
7
2
9
8
1

union(Intersect(A, B), Intersect(A, C), Intersect(B, C))

[143] ✓ 0.0s
... Set{Int64} with 6 elements:
0
4
7
9
3
1
```

2. Приведем свои примеры с выполнением операций над множествами элементов разных типов. (рис. (fig:016?) - (fig:017?))

Задание 2

Приведите свои примеры с выполнением операций над множествами элементов разных типов.

```
S1 = Set([10, 2.0, 3, 4])
S2 = Set(["Some", "text", "and", 10])

println("Значения множества S1: ")
for i in S1
    println(i)
end

println("\nЗначения множества S2: ")
for i in S2
    println(i)
end

println("\nРавенство множеств:")
println(isequal(S1, S2))

println("\nПересечение множеств:")
println(intersect(S1, S2))

println("\nОбъединение множеств:")
println(union(S1, S2))

println("\nРазность множеств:")
println(setdiff(S1, S2))

println("\nВхождение элементов одного множества в другое:")
println(issubset(S1, S2))
```

[144] ✓ 0/0

Рис. 15: Задание 2

## Выполнение заданий для самостоятельной работы

- Приведем свои примеры с выполнением операций над множествами элементов разных типов

```
Элементы множества S1:  
4.0  
2.0  
10.0  
3.0  
  
Элементы множества S2:  
10  
Some  
text  
and  
  
Проверка эквивалентности:  
false  
  
Пересечение множеств:  
Set[Any](10)  
  
Объединение множеств:  
Set[Any](4.0, 2.0, 10, "Some", "text", 3.0, "and")  
  
Разность множеств:  
Set[Any](4.0, 2.0, 10, "Some", "text", 3.0, "and")  
  
Проверка вхождения элементов одного множества в другое:  
false
```

Рис. 16: Задание 2

## Выполнение заданий для самостоятельной работы

- Приведем свои примеры с выполнением операций над множествами элементов разных типов

```
println("\nдобавление элемента в множество S2:")
push!(S2, "end")

println("\nэлементы множества S2: ")
for i in S2
    println(i)
end

println("\nудаление последнего элемента в множестве S2:")
pop!(S2)

println("\nэлементы множества S2: ")
for i in S2
    println(i)
end
```

✓ 0.0s

добавление элемента в множество S2:

Элементы множества S2:

10  
Some  
text  
and  
end

удаление последнего элемента в множестве S2:

Элементы множества S2:

Some  
text  
and  
end

1. Создадим разными способами массивы и вектора. Для создания нужных массивов, используем генераторы и циклы(рис. (fig:018?) - (fig:023?))

[illegible]

Рис. 18: Задание 3

# Выполнение заданий для самостоятельной работы

вектор значений  $y = e^x \cos(x)$  в точках  $x = 3, 3.1, 3.2, \dots, 6$ , найдите среднее значение  $y$ ;

```
function Y(x)
    exp(x)*cos(x)
end
[168] ✓ 0.2s

... Y (generic function with 1 method)

X = 3:0.1:6
res = [Y(x) for x in X]
println(sum(res)/sizeof(res))
```

```
[178] ✓ 0.0s

... 6.639218243303714
```

вектор вида  $(x_i, y_j)$ ,  $x = 0.1, i = 3, 6, 9, \dots, 36, y = 0.2, j = 1, 4, 7, \dots, 34$ ;

```
is = 3:3:36
js = 1:3:34
x = 0.1
y = 0.2
v = [ [x^is[k], y^js[j]] for k in 1:length(is)]
```

```
[182] ✓ 0.6s

... 12-element Vector{Vector{Float64}}:
 [0.0010000000000000002, 0.2]
 [1.0000000000000004e-6, 0.0016000000000000003]
 [1.0000000000000005e-9, 1.2800000000000005e-5]
 [1.0000000000000006e-12, 1.0240000000000006e-7]
 [1.0000000000000009e-15, 8.1920000000000005e-10]
 [1.000000000000001e-18, 6.5536000000000005e-12]
 [1.0000000000000012e-21, 5.2428800000000005e-14]
 [1.0000000000000014e-24, 4.1943040000000005e-16]
 [1.0000000000000015e-27, 3.3544320000000004e-18]
 [1.0000000000000017e-30, 2.6843545600000004e-20]
 [1.0000000000000018e-33, 2.14748364800000035e-22]
 [1.000000000000002e-36, 1.7179869184000003e-24]
```



## Выполнение заданий для самостоятельной работы

вектор с элементами  $2^i i$ ,  $i = 1, 2, \dots, M$ ,  $M = 25$ ;

```
M = 25
is = 1:M
v1 = [2^i/i for i in is]
```

184] ✓ 0.0s

... 25-element Vector{Float64}:

```
 2.0
 2.0
 2.6666666666666665
 4.0
 6.4
10.666666666666666
18.285714285714285
32.0
56.888888888888886
102.4
 ⋮
7710.117647058823
14563.555555555555
27594.105263157893
52428.8
99864.38095238095
190650.18181818182
364722.0869565217
699050.6666666666
1.34217728e6
```

вектор вида ("fn1", "fn2", ..., "fnN"),  $N = 30$ ;

```
v2 = []
for i=1:30
    push!(v2, "fn"*string(i))
end
v2
```

187] ✓ 0.4s

... 30-element Vector{Any}:

```
"fn1"
"fn2"
```

## Выполнение заданий для самостоятельной работы

```
▷ v2 = []  
  for i=1:30  
    push!(v2, "fn"*string(i))  
  end  
  v2
```

[187] ✓ 0.4s

```
... 30-element Vector{Any}:  
  "fn1"  
  "fn2"  
  "fn3"  
  "fn4"  
  "fn5"  
  "fn6"  
  "fn7"  
  "fn8"  
  "fn9"  
  "fn10"  
  ⋮  
  "fn22"  
  "fn23"  
  "fn24"  
  "fn25"  
  "fn26"  
  "fn27"  
  "fn28"  
  "fn29"  
  "fn30"
```

## Выполнение заданий для самостоятельной работы

```
3.14

n = 250
x = [rand(0:999) for i=1:n]
y = [rand(0:999) for i=1:n]
import Plots
fig = plot("Statistics")
using Statistics

Resolving package version...
Updating "C:\Users\UPK\julia\environments\v1.10\Project.toml"
[10745181] + Statistics v1.10.0
No changes to "C:\Users\UPK\julia\environments\v1.10\Manifest.toml"

arr1 = [y[i+1] - x[i] for i=1:n-1]
println("3.14 1) ", arr1)
arr2 = [x[i] + 2*x[i+1] - x[i+2] for i=1:n-2]
println("3.14 2) ", arr2)
arr3 = [sin(x[i])/cos(x[i+1]) for i=1:n-1]
println("3.14 3) ", arr3)
arr4 = sum([exp(-x[i])/x[i] + 10) for i=1:n-1])
println("3.14 4) ", arr4)

3.14 1) [81, 103, -532, -699, -383, -111, -59, 362, 668, 12, -46, 363, 44, -13, 115, -23, 365, 139, -20, 91, 223, 656, 166, -297, -272, -121, -131, -629, 386, 314, 184, -521, 566, 64, -453, 229, -125, 216, -391, 439, -229, -13.14 2) [1469, 1873, 2226, 1686, 417, 1371, 882, -121, 758, 2349, 949, 1228, 1530, 2373, 532, 466, 1488, 1051, 2317, 833, -531, 1173, 999, 581, 1689, 1761, 2304, 959, 597, 236, 1893, 1151, -324, 1180, 18, 964, 1484, 1335, 310 3.14 3) [4.318362966951079, 1.4896581241131672, -1.1043898332335362, -1.3286156449152664, 0.0913924611411679, 0.5845285799115562, 0.7114443231417061, 16.49858267092175, -32.50004518492681, -1.2722173668189565, -1.14372764070 3.14 4) 0.001411189507874082
```

Рис. 22: Задание 3

# Выполнение заданий для самостоятельной работы

```
arr_over600_inds = [i for i, y in enumerate(arr) if y > 600]
arr_over600 = [x for x, i in enumerate(arr) if i in arr_over600_inds]
println("3.14 5) ", arr_over600)
arr_over600_s = [a[i] for i in arr_over600_inds]
println("3.14 6) ", arr_over600_s)
mean = mean(x)
vx = [abs(x[i] - mean)*0.5 for i in range(1, len(arr))]
println("3.14 7) ", vx)
elisy = sum(vx, start=0)
println("3.14 8) ", elisy)
x_odd = sum(map(lambda x: x%2, arr))
x_even = sum(map(lambda x: x%2, arr))
println("3.14 9) ", x_odd, x_even)
x7 = sum([1 for i in range(1, len(arr)) if x[i]%7==0])
println("3.14 10) ", x7)
xsorry = x[-sort(arr)]
println("3.14 11) ", xsorry)
xstop10 = last(sort(arr), 10)
println("3.14 12) ", xstop10)
xunique = collect(x)
println("3.14 13) ", xunique)
```

3.14 5) [973, 913, 989, 688, 777, 938, 747, 856, 771, 905, 985, 864, 929, 743, 912, 822, 708, 821, 677, 747, 616, 788, 644, 852, 845, 834, 828, 734, 626, 978, 684, 917, 912, 945, 677, 982, 853, 842, 789, 678, 982, 898, 651, 1

3.14 6) [838, 886, 973, 705, 984, 384, 812, 784, 858, 111, 725, 582, 41, 746, 486, 835, 852, 365, 364, 66, 574, 481, 583, 391, 155, 743, 668, 488, 357, 529, 311, 805, 968, 98, 37, 394, 76, 985, 968, 972, 138, 271, 174, 739,

3.14 7) [17.629528698811207, 18.9551705488011236, 21.382582883118165, 18.968394746811185, 1.7482387362811886, 15.368799542759585, 11.618460218999992, 17.297198648351724, 17.5271211648165318, 15.678818877812887, 21.5502287051138

3.14 8) 71

3.14 9) 126126

3.14 10) 43

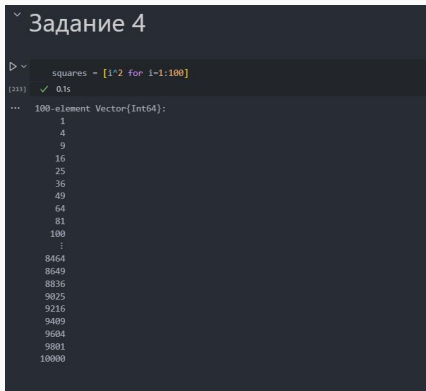
3.14 11) [138, 385, 415, 286, 987, 314, 984, 653, 568, 385, 223, 774, 111, 517, 858, 533, 523, 212, 705, 121, 649, 188, 549, 671, 385, 135, 878, 892, 916, 559, 637, 764, 652, 283, 581, 818, 681, 782, 882, 278, 935, 5, 654, 6

3.14 12) [138, 385, 415, 286, 987, 314, 984, 653, 568, 385, 223, 774, 111, 517, 858, 533, 523, 212, 705, 121, 649, 188, 549, 671, 289, 135, 878, 892, 916, 559, 637, 764, 652, 283, 581, 818, 681, 782, 882, 228, 935, 5, 654, 6

3.14 13) [838, 886, 973, 705, 984, 384, 812, 784, 858, 111, 548, 725, 582, 838, 41, 87, 746, 486, 559, 943, 815, 852, 235, 363, 364, 855, 181, 66, 637, 68, 739, 576, 485, 285, 385, 988, 984, 671

Рис. 23: Задание 3

4. Создадим массив `squares`, в котором будут храниться квадраты всех целых чисел от 1 до 100.(рис. (fig:024?) )



The screenshot shows a code editor with a dark theme. At the top, a tab is labeled "Задание 4". Below the tab, a code line is highlighted: `squares = [i^2 for i=1:100]`. Below this, a status bar indicates the array has 100 elements and is of type Vector{Int64}. The array's contents are displayed as a list of squares from 1 to 100, with an ellipsis indicating the middle elements.

```
Задание 4
squares = [i^2 for i=1:100]
(100) 0.1s
... 100-element Vector{Int64}:
 1
 4
 9
16
25
36
49
64
81
100
 ⋮
8464
8649
8836
9025
9216
9409
9604
9801
10000
```

Рис. 24: Задание 4

## Выполнение заданий для самостоятельной работы

5. Подключим пакет Primes (функции для вычисления простых чисел). Затем сгенерируем массив `myprimes`, в котором будут храниться первые 168 простых чисел. Определим также 89-е наименьшее простое число и срез массива с 89-го до 99-го элемента включительно, содержащий наименьшие простые числа.(рис. (fig:025?))

```
Задание 5

Pkg.add("Primes")
import Primes as pm

[215] ✓ 176s

...
Resolving package versions...
Installed IntegerMathUtils - v0.1.2
Installed Primes - v0.5.6
Updating 'C:\Users\HPA\julia\environments\v1.10\Project.toml'
[27ebfcd6] + Primes v0.5.6
Updating 'C:\Users\HPA\julia\environments\v1.10\Manifest.toml'
[18e54da8] + IntegerMathUtils v0.1.2
[27ebfcd6] + Primes v0.5.6
Precompiling project...
✓ IntegerMathUtils
✓ Primes
2 dependencies successfully precompiled in 11 seconds, 315 already precompiled.

myprimes = [pm.prime(i) for i=1:168]
prime_89 = myprimes[89]
prime_89_to_99 = myprimes[89:99]

[217] ✓ 00s

...
11-element Vector{Int64}:
461
463
467
479
487
491
499
503
509
521
```

## 6. Вычислим выражения(рис. (fig:026?) )

Задание 6

```
# 6.1
res61 = sum([i^3 + 4*i^2 for i=10:100])
println("6.1) ", res61)

# 6.2
res62 = sum([ (2^i)/(i) + (3^i)/(i^2) ] for i=1:25)
println("6.2) ", res62)

# 6.3
res63 = 1
tmp = 1
for i=1:2:38
    tmp*= i/(i+1)
    res63 = res63 + tmp
end
println("6.3) ", res63)
```

219] ✓ 0.1s

... 6.1) 26852735  
6.2) 2.1291704368143802e9  
6.3) 5.01482750478317

Рис. 26: Задание 6

## Выводы

---



- В результате выполнения данной лабораторной работы были изучены структуры данных, реализованных в Julia: словарь, массив, кортеж множество, также были получены практические навыки применения этих структур и операций над ними решения задач.