

# **Лабораторная работа № 2**

**Структуры данных**

Оразгелдиев Язгелди

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Выполнение лабораторной работы</b>	<b>7</b>
<b>4</b>	<b>Выводы</b>	<b>18</b>

# Список иллюстраций

3.1	Примеры использования кортежей . . . . .	7
3.2	Примеры использования кортежей . . . . .	8
3.3	Примеры использования словарей . . . . .	8
3.4	Примеры использования множеств . . . . .	9
3.5	Примеры использования массивов . . . . .	10
3.6	Примеры использования массивов . . . . .	10
3.7	Примеры использования массивов . . . . .	11
3.8	Задание №1. Работа с множествами . . . . .	11
3.9	Задание №2. Примеры операций над множествами элементов раз- ных типов . . . . .	12
3.10	Задание №3. Работа с массивами . . . . .	12
3.11	Задание №3. Работа с массивами . . . . .	12
3.12	Задание №3. Работа с массивами . . . . .	13
3.13	Задание №3. Работа с массивами . . . . .	13
3.14	Задание №3. Работа с векторами . . . . .	14
3.15	Задание №3. Работа с векторами . . . . .	14
3.16	Задание №3. Работа с векторами . . . . .	15
3.17	Задание №3. Работа с векторами . . . . .	16
3.18	Задание №3. Работа с векторами . . . . .	16
3.19	Задание №4, №5 и №6 . . . . .	17

## **Список таблиц**

# 1 Цель работы

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

## 2 Задание

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

## 3 Выполнение лабораторной работы

Для начала выполним примеры из раздела про кортежи. Кортеж (Tuple) – структура данных (контейнер) в виде неизменяемой индексируемой последовательности элементов какого-либо типа (элементы индексируются с единицы).

```
[1]: # пустой кортеж:
    ()
[1]: ()

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

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

[4]: # кортеж из элементов разных типов:
    x2 = (1, 2.0, "tmp")
[4]: (1, 2.0, "tmp")

[5]: # именованный кортеж:
    x3 = (a=2, b=1+2)
[5]: (a = 2, b = 3)
```

Рисунок 3.1: Примеры использования кортежей

```
[6]: # длина кортежа x2:
length(x2)

[6]: 3

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

[7]: (1, 2.0, "tmp")

[8]: # произвести какую-либо операцию (сложение)
# с вторым и третьим элементами кортежа x1:
c = x1[1] + x1[3]

[8]: 5

[9]: # обращение к элементам именованного кортежа x3:
x3.a, x3.b, x3.c

[9]: (2, 3, 3)

[10]: # проверка вхождения элементов tmp и 0 в кортеж x2
# (оба способа обращения к методу in()):
in("tmp", x2), 0 in x2

[10]: (true, false)
```

Рисунок 3.2: Примеры использования кортежей

Теперь выполним примеры из раздела про словари. Словарь – неупорядоченный набор связанных между собой по ключу данных.

```
[11]: # создать словарь с именем phonebook:
phonebook = Dict{String, Any} => {"867-5309", "333-5544"}, "Бухгалтерия" => "555-2368"}

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

[12]: # вывести ключи словаря:
keys(phonebook)

[12]: KeySet for a Dict{String, Any} with 2 entries. Keys:
"Бухгалтерия"
"Иванов И.И."

[13]: # вывести значения элементов словаря:
values(phonebook)

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

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

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

[15]: # проверка вхождения ключа в словарь:
haskey(phonebook, "Иванов И.И.")

[15]: true

[16]: # добавить элемент в словарь:
phonebook["Сидоров П.С."] = "555-3344"

[16]: "555-3344"

[17]: # удалить ключ и связанное с ним значение из словаря
pop!(phonebook, "Иванов И.И.")

[17]: {"867-5309", "333-5544"}

[18]: # объединение словарей (функция merge()):
a = Dict{String, Any} => {"foo" => 0.0, "bar" => 42.0};
b = Dict{String, Any} => {"bar" => 17, "baz" => 13.0};
merge(a, b), merge(b, a)

[18]: (Dict{String, Any}{"bar" => 13.0, "baz" => 17, "foo" => 0.0}, Dict{String, Any}{"bar" => 42.0, "baz" => 17, "foo" => 0.0})
```

Рисунок 3.3: Примеры использования словарей

Выполним примеры из раздела про множества. Множество, как структура данных в Julia, соответствует множеству, как математическому объекту, то есть является неупорядоченной совокупностью элементов какого-либо типа.



Возможные операции над множествами: объединение, пересечение, разность; принадлежность элемента множеству.

```
[20]: # создаем множество из четырех целочисленных значений:
A = Set([1, 3, 4, 5])

[20]: Set{Int64} with 4 elements:
5
4
3
1

[21]: # создаем множество из 11 символьных значений:
B = Set("abracadabra")

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

[22]: # проверка эквивалентности двух множеств:
S1 = Set([1, 2]);
S2 = Set([1, 2]);
issetequal(S1, S2)
S3 = Set([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]);
S4 = Set([1, 2, 3]);
issetequal(S3, S4)

[22]: true

[23]: # объединение множеств:
C = union(S1, S2)

[24]: # пересечение множеств:
D = intersect(S1, S3)

[24]: Set{Int64} with 2 elements:
2
1

[25]: # разность множеств:
E = setdiff(S3, S1)

[25]: Set{Int64} with 1 element:
3

[26]: # проверка вхождения элементов одного множества в другое:
issubset(S1, S4)

[26]: true

[27]: # добавление элемента в множество:
push!(S4, 99)
```

Рисунок 3.4: Примеры использования множеств

Выполним примеры из раздела про массивы. Массив — коллекция упорядоченных элементов, размещённая в многомерной сетке. Векторы и матрицы являются частными случаями массивов.

```

empty_array_1 = []
Any[]

# создание пустого массива с конкретным типом:
empty_array_2 = (Int64[])
empty_array_3 = (Float64[])

Float64[]

# вектор-столбец:
a = [1, 2, 3]

3-element Vector{Int64}:
 1
 2
 3

# вектор-строка:
b = [1 2 3]

1x3 Matrix{Int64}:
 1 2 3

# 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

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

1x8 Matrix{Float64}:
 0.37157  0.868185  0.780679  0.952377  _  0.383572  0.442995  0.450085

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

# трёхмерный массив:
D = rand(4, 3, 2)

4x3x2 Array{Float64, 3}:
[:, :, 1] =
 0.101011  0.862076  0.188035
 0.691591  0.808666  0.505946
 0.307612  0.0336982  0.846561
 0.287509  0.399469  0.996038

[:, :, 2] =
 0.84059  0.697702  0.0540585
 0.673059  0.447209  0.231141
 0.207027  0.224459  0.735297
 0.809726  0.465222  0.481809

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

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^i*i^2,
# где i - нечётное число от 1 до 9 (включительно)
ar_1 = [3^i*i^2 for i in 1:2:9]

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

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

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

```

Рисунок 3.5: Примеры использования массивов

```

# трёхмерный массив:
D = rand(4, 3, 2)

4x3x2 Array{Float64, 3}:
[:, :, 1] =
 0.101011  0.862076  0.188035
 0.691591  0.808666  0.505946
 0.307612  0.0336982  0.846561
 0.287509  0.399469  0.996038

[:, :, 2] =
 0.84059  0.697702  0.0540585
 0.673059  0.447209  0.231141
 0.207027  0.224459  0.735297
 0.809726  0.465222  0.481809

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

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^i*i^2,
# где i - нечётное число от 1 до 9 (включительно)
ar_1 = [3^i*i^2 for i in 1:2:9]

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

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

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

```

Рисунок 3.6: Примеры использования массивов

```

# одномерный массив из пяти единиц:
ones(5)

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

# двумерный массив 2х3 из единиц:
ones(2,3)

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

# одномерный массив из 4 нулей:
zeros(4)

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

# заполнить массив 3х2 цифрами 3.5
fill(3.5,(3,2))

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

# заполнить массива посредством функции repeat():
repeat([1,2],3,2)
repeat([1 2],3,2)

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

```

Рисунок 3.7: Примеры использования массивов

Перейдем к выполнению заданий. 1. Даны множества:  $A = \{0, 3, 4, 9\}$ ,  $B = \{1, 3, 4, 7\}$ ,  $C = \{0, 1, 2, 4, 7, 8, 9\}$ . Найдем  $A \cap B \cup A \cap C \cup B \cap C \cup A \cap B \cap C$

```

# Выполнение самостоятельной работы:

# Задание №1.

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, C), intersect(B, C)) # union(intersect(A, B), intersect(A, B)) = intersect(A, B)

println(P)

Set{[0, 4, 7, 9, 3, 1]}

```

Рисунок 3.8: Задание №1. Работа с множествами

Приведем свои примеры с выполнением операций над множествами элементов разных типов



```
# Пункт 3.11. Вектор вида (xi, yj), x = 0.1, i = 3, 6, 9, ..., 36, y = 0.2, j = 1, 4, 7, ..., 34.
x_vector = 0.1 * collect(3:3:36)
y_vector = 0.2 * collect(1:3:34)
xy_pairs = [(x, y) for x in x_vector, y in y_vector]

12x12 Matrix{Tuple{Float64, Float64}}:
(0.3, 0.2) (0.3, 0.8) (0.3, 1.4) ... (0.3, 5.6) (0.3, 6.2) (0.3, 6.8)
(0.6, 0.2) (0.6, 0.8) (0.6, 1.4) ... (0.6, 5.6) (0.6, 6.2) (0.6, 6.8)
(0.9, 0.2) (0.9, 0.8) (0.9, 1.4) ... (0.9, 5.6) (0.9, 6.2) (0.9, 6.8)
(1.2, 0.2) (1.2, 0.8) (1.2, 1.4) ... (1.2, 5.6) (1.2, 6.2) (1.2, 6.8)
(1.5, 0.2) (1.5, 0.8) (1.5, 1.4) ... (1.5, 5.6) (1.5, 6.2) (1.5, 6.8)
(1.8, 0.2) (1.8, 0.8) (1.8, 1.4) ... (1.8, 5.6) (1.8, 6.2) (1.8, 6.8)
(2.1, 0.2) (2.1, 0.8) (2.1, 1.4) ... (2.1, 5.6) (2.1, 6.2) (2.1, 6.8)
(2.4, 0.2) (2.4, 0.8) (2.4, 1.4) ... (2.4, 5.6) (2.4, 6.2) (2.4, 6.8)
(2.7, 0.2) (2.7, 0.8) (2.7, 1.4) ... (2.7, 5.6) (2.7, 6.2) (2.7, 6.8)
(3.0, 0.2) (3.0, 0.8) (3.0, 1.4) ... (3.0, 5.6) (3.0, 6.2) (3.0, 6.8)
(3.3, 0.2) (3.3, 0.8) (3.3, 1.4) ... (3.3, 5.6) (3.3, 6.2) (3.3, 6.8)
(3.6, 0.2) (3.6, 0.8) (3.6, 1.4) ... (3.6, 5.6) (3.6, 6.2) (3.6, 6.8)

# Пункт 3.12. Вектор с элементами 2^(i)/i, i = 1, 2, ..., M, M = 25.
M = 25
arr12 = [2^i / i for i in 1:M]

25-element Vector{Float64}:
 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
1120.2053142857143
```

Рисунок 3.12: Задание №3. Работа с массивами

```
# Пункт 3.13. Вектор вида ("fn1", "fn2", ..., "fnN"), N = 30.
N = 30
arr13 = ["fn$i" for i in 1:N]

30-element Vector{String}:
"fn1"
"fn2"
"fn3"
"fn4"
"fn5"
"fn6"
"fn7"
"fn8"
"fn9"
"fn10"
"fn11"
"fn12"
"fn13"
⋮
"fn19"
"fn20"
"fn21"
"fn22"
"fn23"
"fn24"
"fn25"
"fn26"
"fn27"
"fn28"
"fn29"
"fn30"
```

Рисунок 3.13: Задание №3. Работа с массивами

```
# Пункт 3.14.
# Создаем векторы x и y:
n = 250
x = rand(0:999, n)
y = rand(0:999, n)
vec1 = y[2:end] - x[1:end-1] # Вектор (y2 - x1, ..., yn - xn-1).
```

249-element Vector{Int64}:

```
285
-40
-16
507
-520
31
-498
865
342
439
213
-520
86
⋮
153
407
641
-495
490
504
575
-113
508
-362
674
-106
```

Рисунок 3.14: Задание №3. Работа с векторами

```
vec2 = x[1:end-2] + 2*x[2:end-1] - x[3:end] # Вектор (x1 + 2x2 - x3, ..., xn-2 + 2xn-1 - xn).
```

248-element Vector{Int64}:

```
1788
1310
-197
1010
619
1392
167
670
595
-249
1494
679
371
⋮
1764
1675
98
1152
1270
44
-145
1989
35
1668
1023
577
```

```
vec3 = sin.(y[1:end-1]) ./ cos.(x[2:end]) # Вектор (sin(y1)/cos(x2), sin(y2)/cos(x3), ..., sin(yn-1)/cos(xn)).
```

249-element Vector{Float64}:

```
71.10216198396886
0.47554489222739715
-29.6898984178015
0.48649014454320944
-1.9703028726577705
2.5620313444536262
-187.9458286643168
0.6484832439940297
-0.4943483013699883
0.8733695764118081
-3.3861450041093057
```

Рисунок 3.15: Задание №3. Работа с векторами

```

sum_expr = sum(exp.(-x[2:end])) .* x[1:end-1] .+ 10) # Вычисление суммы.
println(sum_expr)
y_gt_600 = y[y .> 600]
indices_y_gt_600 = findall(y .> 600) # Элементы y > 600.
println(indices_y_gt_600)

2585.7251975939917
60, 61, 63, 68, 70, 71, 76, 77, 78, 80, 82, 83, 84, 85, 87, 88, 96, 97, 99, 100, 101, 102, 108, 109, 11
0, 112, 114, 117, 121, 124, 125, 127, 128, 132, 134, 138, 140, 143, 144, 148, 149, 153, 157, 158, 162, 16
4, 167, 169, 172, 173, 175, 176, 180, 184, 186, 188, 190, 191, 193, 195, 196, 197, 198, 200, 201, 203, 20
4, 205, 208, 212, 214, 216, 222, 223, 224, 226, 227, 228, 229, 233, 237, 239, 240, 241, 243, 245, 246, 24
9]

x_for_y_gt_600 = x[y .> 600] # Соответствующие значения x для y > 600.
println(x_for_y_gt_600)

[265, 635, 314, 309, 138, 212, 857, 501, 889, 18, 6, 516, 148, 112, 982, 231, 156, 32, 659, 304, 373, 87,
928, 939, 446, 537, 885, 157, 413, 166, 153, 748, 387, 893, 164, 287, 550, 143, 501, 15, 476, 593, 294, 1
92, 379, 224, 560, 125, 537, 793, 38, 102, 749, 337, 534, 730, 96, 536, 626, 444, 457, 800, 581, 254, 90
0, 570, 291, 119, 358, 868, 377, 987, 584, 855, 145, 554, 17, 99, 662, 62, 514, 665, 992, 152, 9, 736, 22
8, 678, 703, 694, 347, 738, 27, 326, 46, 790, 711, 445, 996, 615, 797, 197, 88, 120, 922]

vec4 = abs.(x .- x_mean) .^ 12 # Вектор  $(|x_1 - \bar{x}|^{12}, |x_2 - \bar{x}|^{12}, \dots, |x_n - \bar{x}|^{12})$ .

250-element Vector{Float64}:
 1.626841250961042e28
 1.2733281030630335e20
 8.391475832166273e29
 9.106847060602672e25
 5.96356875145031e27
 8.412620131726251e22
 1.8190704970932972e29
 8.456028570358172e26
 1.184902783279825e27
 1.1294437507797768e28
 1.041416965953231e22
 3.540200191266548e30
 2.0726498713476427e29
 ⋮
 5.80628634328538e19
 5.673710671956571e29
 2.860182204554349e32
 1.8700882101472678e26
 1.5471596580244504e25

```

Рисунок 3.16: Задание №3. Работа с векторами

```
count_y_within_200 = count(y -> y >= maximum(y) - 200, y) # Элементы y, отстоящие от максимума не более
println(count_y_within_200)
even_count = count(iseven, x) # Количество четных и нечетных элементов в x.
odd_count = count(isodd, x)
println(even_count, odd_count)
multiples_of_7 = count(x -> x % 7 == 0, x) # Элементы x, кратные 7.
println(multiples_of_7)
```

250  
140110  
39

```
sorted_x_by_y = x[sortperm(y)] # Сортировка x по возрастанию y.
```

250-element Vector{Int64}:  
172  
129  
884  
386  
941  
849  
866  
74  
762  
801  
22  
644  
423  
:  
928  
939  
992  
294  
987  
291  
102  
748  
790  
152  
537  
445

Рисунок 3.17: Задание №3. Работа с векторами

```
top_10_x = sort(x, rev=true)[1:10] # Топ-10 наибольших элементов x.
```

10-element Vector{Int64}:  
996  
992  
987  
982  
982  
969  
965  
965  
958  
951

```
unique_x = unique(x) # Уникальные элементы в x.
```

223-element Vector{Int64}:  
265  
442  
801  
635  
283  
408  
215  
314  
309  
707  
421  
138  
212  
:  
531  
815  
445  
791  
996  
644

Рисунок 3.18: Задание №3. Работа с векторами



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

Подключим пакет Primes (функции для вычисления простых чисел). Сгенерируем массив myprimes, 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

```
# Пункт 4. Создание массива squares.
squares = [i^2 for i in 1:100]
println(squares)
# Пункт 5. Работа с пакетом Primes.
using Primes
# Получаем первые 168 простых чисел
myprimes = primes(2, 1000) # Предполагаем, что первые 168 простых чисел находятся в этом диапазоне
eighty_ninth_prime = myprimes[89]
slice_89_to_99 = myprimes[89:99]
println(myprimes)
println(eighty_ninth_prime)
println(slice_89_to_99)

[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, 1369, 1444, 1521, 1600, 1681, 1764, 1849, 1936, 2025, 2116, 2209, 2304, 2401, 2500, 2601, 2704, 2809, 2916, 3025, 3136, 3249, 3364, 3481, 3600, 3721, 3844, 3969, 4096, 4225, 4356, 4489, 4624, 4761, 4900, 5041, 5184, 5329, 5476, 5625, 5776, 5929, 6084, 6241, 6400, 6561, 6724, 6889, 7056, 7225, 7396, 7569, 7744, 7921, 8100, 8281, 8464, 8649, 8836, 9025, 9216, 9409, 9604, 9801, 10000]
[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, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557, 563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647, 653, 659, 661, 673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773, 787, 797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881, 883, 887, 907, 911, 919, 929, 937, 941, 947, 953, 967, 971, 977, 983, 991, 997]
461
[461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523]
```

```
# Пункт 6. Вычисление выражений.
sum_expr_6_1 = sum([i^3 + 4*i^2 for i in 10:100]) # Первая сумма
println(sum_expr_6_1)
M = 25
sum_expr_6_2 = sum([2 + 3/i for i in 1:M]) # Вторая сумма
println(sum_expr_6_2)
sum_expr_6_3 = 1 + sum([prod(2:i)/prod(3:i+1) for i in 2:38]) # Третья сумма
println(sum_expr_6_3)

26852735
61.44787453326052
12.290893162283911
```

Рисунок 3.19: Задание №4, №5 и №6

## 4 Выводы

Мы изучили несколько структур данных, реализованных в Julia, научились применять их и операции над ними для решения задач.