Лабораторная работа № 2. Структуры данных

Адабор Кристофер Твум

1032225824

НКНбд-01-22

Цель работы

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

Задание(повторяем примеры из раздела).

In [7]:
```
# Кортежи:

# пустой кортеж:
()
```

Out[7]:
```
()
```

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

Out[2]:
```
("Python", "Julia", "R")
```

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

Out[3]:
```
(1, 2, 3)
```

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

Out[4]:
```
(1, 2.0, "tmp")
```

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

Out[5]:
```
(a = 2, b = 3)
```

In [8]:
```
# Примеры операций над кортежами:

# длина кортежа x2:
length(x2)
```

Out[8]: 3

In [11]:
```
# обратиться к элементам кортежа x2:
x2[1], x2[2], x2[3]
```

Out[11]: (1, 2.0, "tmp")

In [12]:
```
# произвести какую-либо операцию (сложение)
# с вторым и третьим элементами кортежа x1:
c = x1[2] + x1[3]
```

Out[12]: 5

In [13]:
```
# обращение к элементам именованного кортежа x3:
x3.a, x3.b, x3[2]
```

Out[13]: (2, 3, 3)

In [14]:
```
# проверка вхождения элементов tmp и 0 в кортеж x2
# (два способа обращения к методу in()):
in("tmp", x2), 0 in x2
```

Out[14]: (true, false)

In [16]:
```
# Словари:

# создать словарь с именем phonebook:
phonebook = Dict("Иванов И.И." => ("867-5309","333-5544"), "Бухгалтерия" => "555-23
```

Out[16]:
```
Dict{String, Any} with 2 entries:
  "Бухгалтерия" => "555-2368"
  "Иванов И.И." => ("867-5309", "333-5544")
```

In [17]:
```
# вывести ключи словаря:
keys(phonebook)
```

Out[17]:
```
KeySet for a Dict{String, Any} with 2 entries. Keys:
  "Бухгалтерия"
  "Иванов И.И."
```

In [18]:
```
# вывести значения элементов словаря:
values(phonebook)
```

Out[18]:
```
ValueIterator for a Dict{String, Any} with 2 entries. Values:
  "555-2368"
  ("867-5309", "333-5544")
```

In [19]:
```
# вывести заданные в словаре пары "ключ - значение":
pairs(phonebook)
```

```
Out[19]: Dict{String, Any} with 2 entries:
           "Бухгалтерия" => "555-2368"
           "Иванов И.И." => ("867-5309", "333-5544")
```

```
In [20]: # проверка вхождения ключа в словарь:
         haskey(phonebook, "Иванов И.И.")
```

```
Out[20]: true
```

```
In [21]: # добавить элемент в словарь:
         phonebook["Сидоров П.С."] = "555-3344"
```

```
Out[21]: "555-3344"
```

```
In [23]: # удалить ключ и связанные с ним значения из словаря
         pop!(phonebook, "Иванов И.И.")
```

```
Out[23]: ("867-5309", "333-5544")
```

```
In [24]: # 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)
```

```
Out[24]: (Dict{String, Real}("bar" => 13.0, "baz" => 17, "foo" => 0.0), Dict{String, Real}
         ("bar" => 42.0, "baz" => 17, "foo" => 0.0))
```

In [26]:
```
# Множества:

# создать множество из четырёх целочисленных значений:
A = Set([1, 3, 4, 5])
```

Out[26]:
```
Set{Int64} with 4 elements:
  5
  4
  3
  1
```

In [27]:
```
# создать множество из 11 символьных значений:
B = Set("abrakadabra")
```

Out[27]:
```
Set{Char} with 5 elements:
  'a'
  'd'
  'r'
  'k'
  'b'
```

In [28]:
```
# проверка эквивалентности двух множеств:
S1 = Set([1,2]);
S2 = Set([3,4]);
issetequal(S1,S2)
```

Out[28]:
```
false
```

In [29]:
```
S3 = Set([1,2,2,3,1,2,3,2,1]);
S4 = Set([2,3,1]);
issetequal(S3,S4)
```

Out[29]:
```
true
```

In [30]:
```
# объединение множеств:
C=union(S1,S2)
```

Out[30]:
```
Set{Int64} with 4 elements:
  4
  2
  3
  1
```

In [31]:
```
# пересечение множеств:
D = intersect(S1,S3)
```

Out[31]:
```
Set{Int64} with 2 elements:
  2
  1
```

In [32]:
```
# разность множеств:
E = setdiff(S3,S1)
```

Out[32]:
```
Set{Int64} with 1 element:
  3
```

In [33]:
```
# проверка вхождения элементов одного множества в другое:
issubset(S1,S4)
```

Out[33]:
```
true
```

In [34]:
```
# добавление элемента в множество:
push!(S4, 99)
```

Out[34]:
```
Set{Int64} with 4 elements:
  2
  99
  3
  1
```

In [35]:
```
# удаление последнего элемента множества:
pop!(S4)
```

Out[35]:
```
2
```

In [36]:
```
# Массивы:

# создание пустого массива с абстрактным типом:
empty_array_1 = []
```

Out[36]:
```
Any[]
```

In [37]:
```
# создание пустого массива с конкретным типом:
```

```
empty_array_2 = (Int64)[]
empty_array_3 = (Float64)[]
```

Out[37]:
```
Float64[]
```

In [38]:
```
# вектор-столбец:
a = [1, 2, 3]
```

Out[38]:
```
3-element Vector{Int64}:
 1
 2
 3
```

In [39]:
```
# вектор-строка:
b = [1 2 3]
```

Out[39]:
```
1×3 Matrix{Int64}:
 1  2  3
```

In [41]:
```
# многомерные массивы (матрицы):
A = [[1, 2, 3] [4, 5, 6] [7, 8, 9]]
```

Out[41]:
```
3×3 Matrix{Int64}:
 1  4  7
 2  5  8
 3  6  9
```

In [42]:
```
B = [[1 2 3]; [4 5 6]; [7 8 9]]
```

Out[42]:
```
3×3 Matrix{Int64}:
 1  2  3
 4  5  6
 7  8  9
```

In [45]:
```
# одномерный массив из 8 элементов (массив $1 \times 8$)
# со значениями, случайно распределёнными на интервале [0, 1):
c = rand(1,8)
```

Out[45]:
```
1×8 Matrix{Float64}:
 0.887967  0.397398  0.916631  0.193065  …  0.757686  0.541473  0.235496
```

In [48]:
```
# многомерный массив $2 \times 3$ (2 строки, 3 столбца) элементов
# со значениями, случайно распределёнными на интервале [0, 1):
C = rand(2,3)
```

Out[48]:
```
2×3 Matrix{Float64}:
 0.895076  0.851244   0.439861
 0.502467  0.0283441  0.569015
```

In [49]:
```
# трёхмерный массив:
D = rand(4, 3, 2)
```

Out[49]:
```
4×3×2 Array{Float64, 3}:
[:, :, 1] =
 0.364802   0.0102573  0.96375
 0.0554138  0.594815   0.880497
 0.52068    0.558104   0.705446
 0.369265   0.465458   0.210343

[:, :, 2] =
 0.658919  0.516749   0.919094
 0.975825  0.0762784  0.98538
 0.356991  0.984591   0.638042
 0.533648  0.123939   0.419453
```

In [50]:
```
# массив из квадратных корней всех целых чисел от 1 до 10:
roots = [sqrt(i) for i in 1:10]
```

Out[50]:
```
10-element Vector{Float64}:
 1.0
 1.4142135623730951
 1.7320508075688772
 2.0
 2.23606797749979
 2.449489742783178
 2.6457513110645907
 2.8284271247461903
 3.0
 3.1622776601683795
```

In [51]:
```
# массив с элементами вида 3*x^2,
# где x - нечётное число от 1 до 9 (включительно)
ar_1 = [3*i^2 for i in 1:2:9]
```

Out[51]:
```
5-element Vector{Int64}:
   3
  27
  75
 147
 243
```

In [52]:
```
# массив квадратов элементов, если квадрат не делится на 5 или 4:
ar_2=[i^2 for i=1:10 if (i^2%5!=0 && i^2%4!=0)]
```

Out[52]:
```
4-element Vector{Int64}:
  1
  9
 49
 81
```

In [53]:
```
# Некоторые операции для работы с массивами:

# одномерный массив из пяти единиц:
ones(5)
```

```
Out[53]: 5-element Vector{Float64}:
          1.0
          1.0
          1.0
          1.0
          1.0
```

```
In [54]: # двумерный массив 2x3 из единиц:
         ones(2,3)
```

```
Out[54]: 2×3 Matrix{Float64}:
          1.0  1.0  1.0
          1.0  1.0  1.0
```

```
In [55]: # одномерный массив из 4 нулей:
         zeros(4)
```

```
Out[55]: 4-element Vector{Float64}:
          0.0
          0.0
          0.0
          0.0
```

```
In [56]: # заполнить массив 3x2 цифрами 3.5
         fill(3.5,(3,2))
```

```
Out[56]: 3×2 Matrix{Float64}:
          3.5  3.5
          3.5  3.5
          3.5  3.5
```

In [60]:
```
# заполнение массива посредством функции repeat():
repeat([1,2],3,3)
```

Out[60]:
```
6×3 Matrix{Int64}:
 1  1  1
 2  2  2
 1  1  1
 2  2  2
 1  1  1
 2  2  2
```

In [61]:
```
repeat([1 2],3,3)
```

Out[61]:
```
3×6 Matrix{Int64}:
 1  2  1  2  1  2
 1  2  1  2  1  2
 1  2  1  2  1  2
```

In [62]:
```
# преобразование одномерного массива из целых чисел от 1 до 12
# в двумерный массив 2x6
a = collect(1:12)
b = reshape(a,(2,6))
```

Out[62]:
```
2×6 Matrix{Int64}:
 1  3  5  7   9  11
 2  4  6  8  10  12
```

In [63]:
```
# транспонирование
b'
```

Out[63]:
```
6×2 adjoint(::Matrix{Int64}) with eltype Int64:
  1   2
  3   4
  5   6
  7   8
  9  10
 11  12
```

In [64]:
```
# транспонирование
c = transpose(b)
```

Out[64]:
```
6×2 transpose(::Matrix{Int64}) with eltype Int64:
  1   2
  3   4
  5   6
  7   8
  9  10
 11  12
```

In [65]:
```
# массив 10x5 целых чисел в диапазоне [10, 20]:
ar = rand(10:20, 10, 5)
```

Out[65]:
```
10×5 Matrix{Int64}:
 15  18  18  14  13
 16  11  12  19  17
 20  16  19  13  13
 15  20  19  17  16
 20  12  17  14  10
 17  20  14  14  19
 17  10  15  12  12
 19  19  16  11  12
 17  10  16  10  14
 10  20  11  18  17
```

In [66]:
```
# выбор всех значений строки в столбце 2:
ar[:, 2]
```

Out[66]:
```
10-element Vector{Int64}:
 18
 11
 16
 20
 12
 20
 10
 19
 10
 20
```

In [67]:
```
# Выбор всех значений в столбцах 2 и 5:
ar[:, [2, 5]]
```

Out[67]:
```
10×2 Matrix{Int64}:
 18  13
 11  17
 16  13
 20  16
 12  10
 20  19
 10  12
 19  12
 10  14
 20  17
```

In [68]:
```
# все значения строк в столбцах 2, 3 и 4:
ar[:, 2:4]
```

Out[68]:
```
10×3 Matrix{Int64}:
 18  18  14
 11  12  19
 16  19  13
 20  19  17
 12  17  14
 20  14  14
 10  15  12
 19  16  11
 10  16  10
 20  11  18
```

In [69]:
```
# значения в строках 2, 4, 6 и в столбцах 1 и 5:
ar[[2, 4, 6], [1, 5]]
```

Out[69]:
```
3×2 Matrix{Int64}:
 16  17
 15  16
 17  19
```

In [70]:
```
# значения в строке 1 от столбца 3 до последнего столбца:
ar[1, 3:end]
```

Out[70]:
```
3-element Vector{Int64}:
 18
 14
 13
```

In [71]:
```
# сортировка по столбцам:
sort(ar,dims=1)
```

Out[71]:
```
10×5 Matrix{Int64}:
 10  10  11  10  10
 15  10  12  11  12
 15  11  14  12  12
 16  12  15  13  13
 17  16  16  14  13
 17  18  16  14  14
 17  19  17  14  16
 19  20  18  17  17
 20  20  19  18  17
 20  20  19  19  19
```

In [72]:
```
# сортировка по строкам:
sort(ar,dims=2)
```

Out[72]:
```
10×5 Matrix{Int64}:
 13  14  15  18  18
 11  12  16  17  19
 13  13  16  19  20
 15  16  17  19  20
 10  12  14  17  20
 14  14  17  19  20
 10  12  12  15  17
 11  12  16  19  19
 10  10  14  16  17
 10  11  17  18  20
```

In [73]:
```
# поэлементное сравнение с числом
# (результат - массив логических значений):
ar .> 14
```

Out[73]:
```
10×5 BitMatrix:
 1  1  1  0  0
 1  0  0  1  1
 1  1  1  0  0
 1  1  1  1  1
 1  0  1  0  0
 1  1  0  0  1
 1  0  1  0  0
 1  1  1  0  0
 1  0  1  0  0
 0  1  0  1  1
```

In [74]:
```
# возврат индексов элементов массива, удовлетворяющих условию:
findall(ar .> 14)
```

```
Out[74]: 29-element Vector{CartesianIndex{2}}:
          CartesianIndex(1, 1)
          CartesianIndex(2, 1)
          CartesianIndex(3, 1)
          CartesianIndex(4, 1)
          CartesianIndex(5, 1)
          CartesianIndex(6, 1)
          CartesianIndex(7, 1)
          CartesianIndex(8, 1)
          CartesianIndex(9, 1)
          CartesianIndex(1, 2)
          CartesianIndex(3, 2)
          CartesianIndex(4, 2)
          CartesianIndex(6, 2)
          ⋮
          CartesianIndex(4, 3)
          CartesianIndex(5, 3)
          CartesianIndex(7, 3)
          CartesianIndex(8, 3)
          CartesianIndex(9, 3)
          CartesianIndex(2, 4)
          CartesianIndex(4, 4)
          CartesianIndex(10, 4)
          CartesianIndex(2, 5)
          CartesianIndex(4, 5)
          CartesianIndex(6, 5)
          CartesianIndex(10, 5)
```

In [76]:
```julia
# 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

println(P)
```

```
Set([0, 4, 7, 9, 3, 1])
```

In [77]:
```julia
# Задание №2.

# Пример 1:
# Создаем множества
set1 = Set([1, 2, 3, "apple", "banana"])
set2 = Set([2, "banana", "cherry", 4])

# Объединение
union_set = union(set1, set2)

# Пересечение
intersection_set = intersect(set1, set2)
```

```
# Разность
difference_set = setdiff(set1, set2)

println("Union: ", union_set)
println("Intersection: ", intersection_set)
println("Difference: ", difference_set)
```

```
Union: Set(Any[4, 2, "cherry", "banana", 3, 1, "apple"])
Intersection: Set(Any[2, "banana"])
Difference: Set(Any[3, 1, "apple"])
```

78]:
```
# Пример 2:
# Создаем множества
setA = Set([1, (2, 3), (4, 5), "hello"])
setB = Set([(2, 3), "world", 1, 6])

# Объединение
unionAB = union(setA, setB)

# Пересечение
intersectionAB = intersect(setA, setB)

# Разность
differenceAB = setdiff(setA, setB)

println("Union: ", unionAB)
println("Intersection: ", intersectionAB)
println("Difference: ", differenceAB)
```

```
Union: Set(Any[(4, 5), 6, "hello", (2, 3), 1, "world"])
Intersection: Set(Any[(2, 3), 1])
Difference: Set(Any[(4, 5), "hello"])
```

```
# Задание №3.

# Пункт 3.1. Массив от 1 до N.
N = 25
arr1 = collect(1:N)
println(arr1)
# Пункт 3.2. Массив от N до 1.
arr2 = collect(N:-1:1)
println(arr2)
# Пункт 3.3. Массив от 1 до N, затем обратно от N до 1.
arr3 = [collect(1:N); collect(N:-1:1)]
println(arr3)
# Пункт 3.4. Массив tmp = (4, 6, 3).
tmp = [4, 6, 3]
println(tmp)
# Пункт 3.5. Массив, где первый элемент tmp повторяется 10 раз.
arr5 = fill(tmp[1], 10)
println(arr5)
# Пункт 3.6. Массив, где все элементы tmp повторяются 10 раз.
arr6 = repeat(tmp, 10)
println(arr6)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23,
24, 25]
[25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3
, 2, 1]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23,
24, 25, 25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6,
5, 4, 3, 2, 1]
[4, 6, 3]
[4, 4, 4, 4, 4, 4, 4, 4, 4, 4]
[4, 6, 3, 4, 6, 3, 4, 6, 3, 4, 6, 3, 4, 6, 3, 4, 6, 3, 4, 6, 3, 4, 6, 3, 4,
6, 3]
```

86]:
```
# Пункт 3.7. Массив, где первый элемент tmp встречается 11 раз, остальные – 10 раз.
arr7 = [fill(tmp[1], 11); fill(tmp[2], 10); fill(tmp[3], 10)]
println(arr7)
# Пункт 3.8. Массив, где элементы tmp встречаются 10, 20 и 30 раз подряд соответств
arr8 = [fill(tmp[1], 10); fill(tmp[2], 20); fill(tmp[3], 30)]
println(arr8)
# Пункт 3.9. Массив из элементов 2tmp[i], i = 1, 2, 3, элемент 2tmp[3] встречается
arr9 = [2*tmp[1], 2*tmp[2], fill(2*tmp[3], 4)...]
count_6s = count(x -> x == 6, arr9)
println("Number of 6's: ", count_6s)
# Пункт 3.10. Вектор значений y = e^(x)*cos(x) в точках x = 3,3.1,3.2,…,6; среднее
using Statistics
x_values = 3:0.1:6
y_values = [exp(x) * cos(x) for x in x_values]
mean_y = mean(y_values)
println(mean_y)
```

```
[4, 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]
[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, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3]
Number of 6's: 4
53.11374594642971
```

In [87]:
```
# Пункт 3.11. Вектор вида (xi, yj), x = 0.1, i = 3, 6, 9, … , 36, y = 0.2, j = 1, 4
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]
```

Out[87]:

12×12 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
    1170.2857142857142
    2184.5333333333333
    4096.0
    7710.117647058823
   14563.555555555555
   27594.105263157893
   52428.8
   99864.38095238095
  190650.18181818182
  364722.0869565217
  699050.6666666666
       1.34217728e6
```

```
# 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"
```

```
# 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).
```

95]: 249-element Vector{Int64}:
 -127
 -410
  -23
 -545
  -96
   16
  696
  507
  214
 -548
  396
  516
  329
    ⋮
  236
 -216
 -173
 -948
  264
  230
  -31
  527
  728
  403
  -93
 -375

96]:
```
vec2 = x[1:end-2] + 2*x[2:end-1] - x[3:end] #  Вектор (x1 + 2x2 - x3, ..., xn-2 + 2
```

```
[96]: 248-element Vector{Int64}:
          348
         1409
         1788
          793
          884
          524
          534
          225
         1302
         1411
          485
          442
          257
            ⋮
         1894
          861
         1031
         2139
         1669
         1077
         2012
         1103
          253
         -471
          796
         1964
```

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

```
249-element Vector{Float64}:
 -0.38526338525874715
  0.22934257546780393
  0.5982532238690296
 -0.9283098480503401
  0.9735400230947757
 -5.0417167021664 41
  0.11622581572725489
 -0.14307524844351727
  0.8733290197248826
  0.9975280246453233
  0.9649471574609657
 -3.2161680306293183
  0.6406468089502774
  ⋮
  1.7517951330789292
 -1.1013863363487328
 -0.33216127895415953
 -0.7797019332587817
 -1.0520941388181548
 -0.13976606854906637
 -0.14583165918581886
 -0.5218265988400567
  1.2167019783868809
 -1.425338570664749
-17.368692180095376
  0.8273184619648049
```

```
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)
```

```
2492.180414380725
[1, 4, 8, 9, 12, 13, 21, 23, 24, 27, 29, 33, 34, 35, 38, 41, 45, 47, 48, 49, 52, 55,
61, 66, 74, 75, 77, 78, 79, 82, 83, 86, 87, 90, 92, 93, 95, 96, 98, 99, 101, 102, 10
5, 106, 109, 111, 112, 113, 118, 122, 123, 127, 134, 135, 136, 137, 142, 148, 149, 1
53, 155, 156, 157, 158, 159, 161, 163, 166, 168, 172, 173, 176, 178, 179, 180, 182,
184, 185, 186, 190, 191, 194, 197, 199, 200, 208, 210, 211, 213, 214, 217, 226, 227,
228, 229, 230, 233, 239, 241, 243, 244, 245, 246, 247, 249]
```

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

```
[265, 635, 314, 389, 138, 212, 857, 501, 889, 18, 6, 516, 148, 112, 982, 231, 156, 3
2, 659, 304, 373, 87, 928, 939, 446, 537, 885, 157, 413, 166, 153, 748, 387, 893, 16
4, 287, 550, 143, 501, 15, 476, 593, 294, 192, 379, 224, 560, 125, 537, 793, 38, 102
, 749, 337, 534, 730, 96, 536, 626, 444, 457, 800, 581, 254, 900, 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 (|x1 - x̄|¹², |x2 - x̄|¹², ..., |xn - x̄|¹²).
```

```
250-element Vector{Float64}:
 1.626841250961042e28
 1.27332810386303 35e20
 8.3914758321662 73e29
 9.1068470606826 72e25
 5.9635687514503 1e27
 8.4126201317262 51e22
 1.8190704970932 972e29
 8.4560285703581 72e26
 1.1849027832798 25e27
 1.1294437507797 768e28
 1.0414169659532 31e22
 3.5402001912665 48e30
 2.0726498713476 427e29
 ⋮
 5.8062863432853 8e19
 5.6737106719565 71e29
 2.8601822045543 49e32
 1.8700882101472 678e26
 1.5471596580244 504e25
 7.1864371574207 5e29
 3.8995584879884 07e29
 1.7474672943944 017e31
 6.4476037397560 42e30
 7.7675615881750 14e29
 4.3005107648088 49e31
 2.1635122560133 56e27
```

```
count_y_within_200 = count(y -> y >= maximum(y) - 200, y) #  Элементы y, отстоящие
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
```

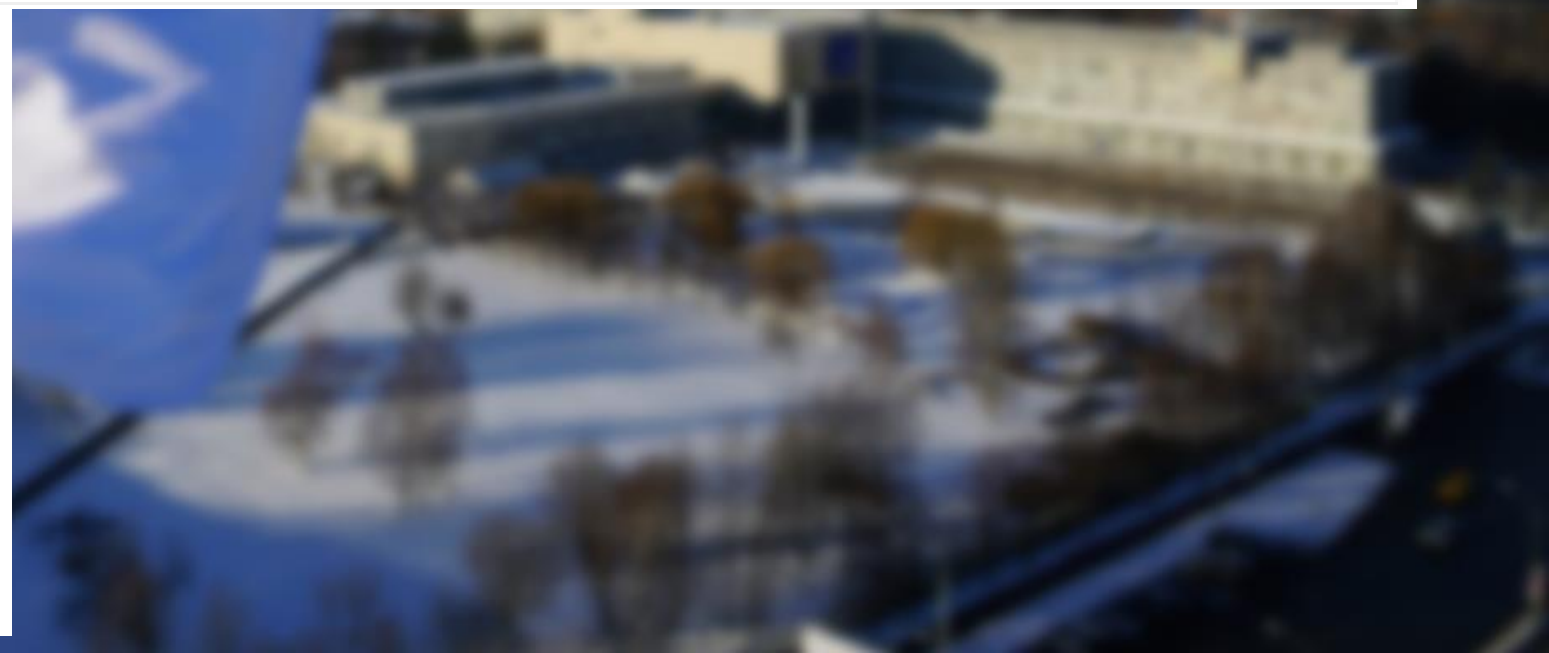
```
top_10_x = sort(x, rev=true)[1:10] #  Топ-10 наибольших элементов x.
```

```
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
 389
 707
 421
 138
 212
   ⋮
 531
 815
 445
 791
 996
 644
 615
 797
 197
 799
 922
 679
```

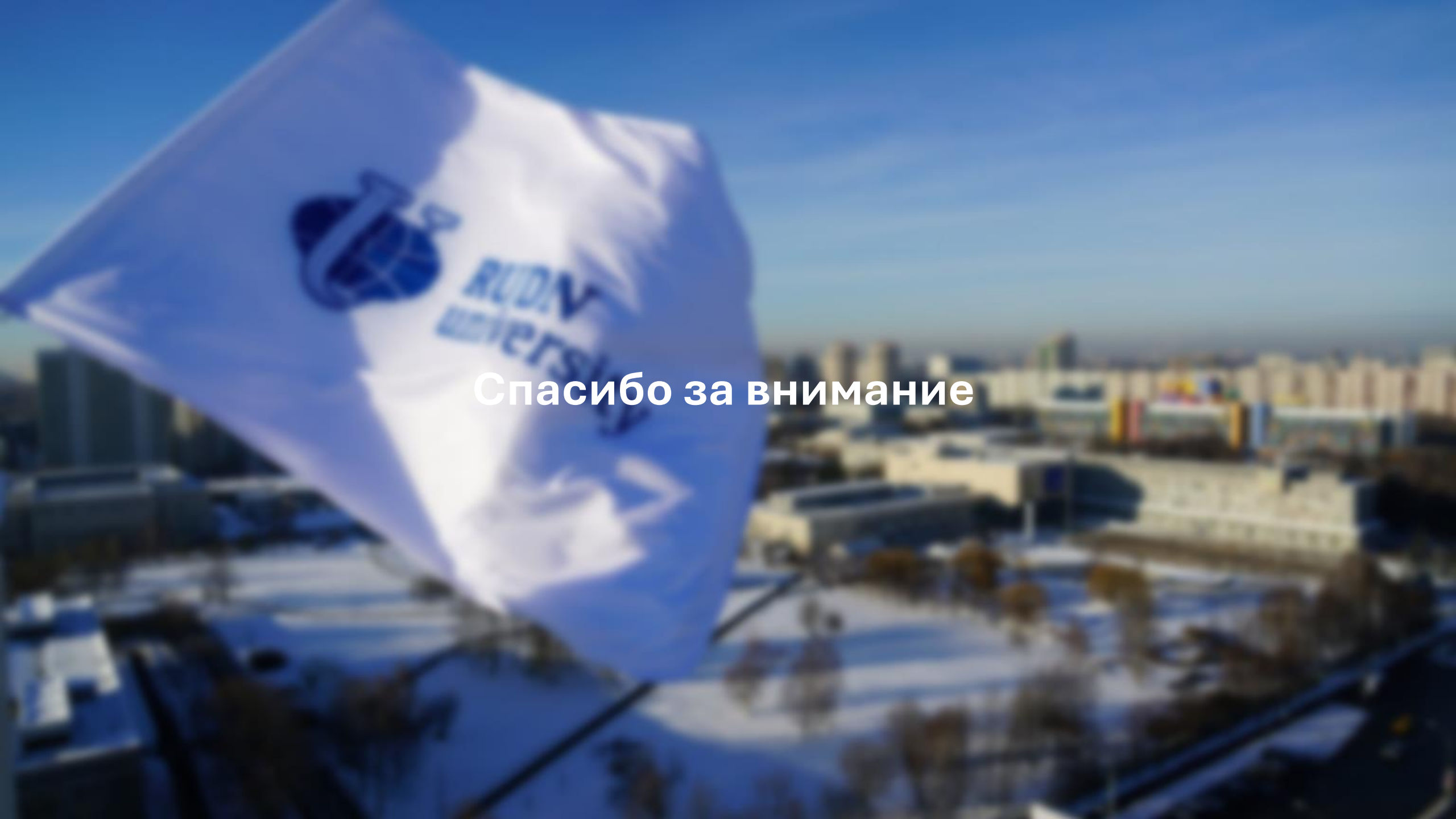
```julia
# Пункт 4. Создание массива squares.
squares = [i^2 for i in 1:100]
println(squares)
# Пункт 5. Работа с пакетом Primes.
using Primes
# Получаем первые 168 простых чисел
myprimes = primes(2, 1000)  # Предполагаем, что первые 168 простых чисел находятся
eighty_ninth_prime = myprimes[89]
slice_89_to_99 = 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, 17
3, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 2
69, 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
```


Спасибо за внимание