

Отчет по лабораторной работе №2

Компьютерный практикум по статистическому анализу данных

Амуничников Антон Игоревич

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	8

Список иллюстраций

4.1	Примеры использования кортежей	8
4.2	Примеры использования словарей	9
4.3	Примеры использования множеств	10
4.4	Примеры использования массивов	11
4.5	Примеры использования массивов	11
4.6	Примеры использования массивов	12
4.7	Примеры использования массивов	13
4.8	Примеры использования массивов	14
4.9	Задание №1. Работа с множествами	15
4.10	Задание №2. Примеры операций над множествами элементов разных типов	16
4.11	Задание №3. Работа с массивами	17
4.12	Задание №3. Работа с массивами	17
4.13	Задание №3. Работа с массивами	18
4.14	Задание №3. Работа с массивами	18
4.15	Задание №3. Работа с массивами	19
4.16	Задание №3. Работа с массивами	19
4.17	Задание №3. Работа с массивами	20
4.18	Задание №4	20
4.19	Задание №5. Работа с пакетом Primes	21
4.20	Задание №6	21

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.
2. Выполните задания для самостоятельной работы.

3 Теоретическое введение

Julia – высокоуровневый свободный язык программирования с динамической типизацией, созданный для математических вычислений **[julia.lang]**. 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 **[julia.doc]**.

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.

4 Выполнение лабораторной работы

Для начала выполним примеры из раздела про кортежи (рис. 4.1). 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)
[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[2] + x1[3]
[8]: 5
[9]: # обращение к элементам именованного кортежа x3:
    x3.a, x3.b, x3[2]
[9]: (2, 3, 3)
[10]: # проверка вхождения элементов tmp и 0 в кортеж x2
    # (два способа обращения к методу in()):
    in("tmp", x2), 0 in x2
[10]: (true, false)
```

Рисунок 4.1: Примеры использования кортежей

Теперь выполним примеры из раздела про словари (рис. 4.2). Словарь – неупорядоченный набор связанных между собой по ключу данных.


```

[12]: # создать словарь с именем phonebook:
phonebook = Dict{String, Any}{"Иванов И.И." => ("867-5309", "333-5544"), "Бухгалтерия" => "555-2368"}

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

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

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

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

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

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

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

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

[16]: true

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

[17]: "555-3344"

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

[19]: ("867-5309", "333-5544")

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

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

```

Рисунок 4.2: Примеры использования словарей

Выполним примеры из раздела про множества (рис. 4.3). 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.

```

[21]: # создать множество из четырёх целочисленных значений:
      A = Set([1, 3, 4, 5])
[21]: Set{Int64} with 4 elements:
      5
      4
      3
      1
[22]: # создать множество из 11 символьных значений:
      B = Set("abracadabra")
[22]: Set{Char} with 5 elements:
      'a'
      'd'
      'r'
      'k'
      'b'
[23]: # проверка эквивалентности двух множеств:
      S1 = Set([1,2]);
      S2 = Set([3,4]);
      issetequal(S1,S2)
      S3 = Set([1,2,2,3,1,2,3,3,1]);
      S4 = Set([2,3,1]);
      issetequal(S3,S4)
[23]: true
[24]: # проверка эквивалентности двух множеств:
      S1 = Set([1,2]);
      S2 = Set([3,4]);
      issetequal(S1,S2)
[24]: false
[25]: C=union(S1,S2)
[25]: Set{Int64} with 4 elements:
      4
      2
      3
      1
[26]: # пересечение множеств:
      D = intersect(S1,S3)
[26]: Set{Int64} with 2 elements:
      2
      1
[27]: # разность множеств:
      E = setdiff(S3,S1)

```

Рисунок 4.3: Примеры использования множеств

Выполним примеры из раздела про массивы (рис. 4.4 - рис. 4.8). Массив — коллекция упорядоченных элементов, размещённая в многомерной сетке. Векторы и матрицы являются частными случаями массивов.

```

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

[31]: Any[]

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

[32]: Float64[]

[33]: # вектор-столбец:
a = [1, 2, 3]

[33]: 3-element Vector{Int64}:
 1
 2
 3

[34]: # вектор-строка:
b = [1 2 3]

[34]: 1x3 Matrix{Int64}:
 1 2 3

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

[35]: 3x3 Matrix{Int64}:
 1 2 3
 4 5 6
 7 8 9

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

[36]: 1x8 Matrix{Float64}:
 0.869786 0.752429 0.401719 0.36821 ... 0.109415 0.367586 0.00477669

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

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

[38]: 4x3x2 Array{Float64, 3}:
[:, :, 1] =
 0.723352 0.00350367 0.729408
 0.237013 0.858544 0.178081
 0.248532 0.808526 0.0206531
 0.436335 0.981371 0.50473

[:, :, 2] =
 0.668084 0.815358 0.439154
 0.525835 0.825208 0.863783
 0.847101 0.310807 0.328788
 0.642303 0.974095 0.305427

```

Рисунок 4.4: Примеры использования массивов

```

[36]: 1x8 Matrix{Float64}:
 0.869786 0.752429 0.401719 0.36821 ... 0.109415 0.367586 0.00477669

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

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

[38]: 4x3x2 Array{Float64, 3}:
[:, :, 1] =
 0.723352 0.00350367 0.729408
 0.237013 0.858544 0.178081
 0.248532 0.808526 0.0206531
 0.436335 0.981371 0.50473

[:, :, 2] =
 0.668084 0.815358 0.439154
 0.525835 0.825208 0.863783
 0.847101 0.310807 0.328788
 0.642303 0.974095 0.305427

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

[39]: 10-element Vector{Float64}:
 1.0
 1.4142135623730951
 1.7320508075688772
 2.0
 2.23606797749979
 2.449489742783178
 2.6457513110645907
 2.8284271247461903
 3.0
 3.1622776601683795

```

Рисунок 4.5: Примеры использования массивов

```

[40]: ar 1 = [3*i^2 for i in 1:2:9]
[40]: 5-element Vector{Int64}:
      3
     27
     75
    147
    243
[41]: # 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)]
[41]: 4-element Vector{Int64}:
      1
      9
     49
     81
[42]: # одномерный массив из пяти единиц:
      ones(5)
[42]: 5-element Vector{Float64}:
     1.0
     1.0
     1.0
     1.0
     1.0
[43]: # двумерный массив 2x3 из единиц:
      ones(2,3)
[43]: 2x3 Matrix{Float64}:
 1.0  1.0  1.0
 1.0  1.0  1.0
[44]: # одномерный массив из 4 нулей:
      zeros(4)
[44]: 4-element Vector{Float64}:
  0.0
  0.0
  0.0
  0.0
[45]: # заполнить массив 3x2 цифрами 3.5
      fill(3.5,(3,2))
[45]: 3x2 Matrix{Float64}:
 3.5  3.5
 3.5  3.5
 3.5  3.5

```

Рисунок 4.6: Примеры использования массивов

```

[48]: # транспонирование
      b'

[48]: 6x2 adjoint(::Matrix{Int64}) with eltype Int64:
      1 2
      3 4
      5 6
      7 8
      9 10
      11 12

[49]: # транспонирование
      c = transpose(b)

[49]: 6x2 transpose(::Matrix{Int64}) with eltype Int64:
      1 2
      3 4
      5 6
      7 8
      9 10
      11 12

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

[50]: 10x5 Matrix{Int64}:
      12 14 14 12 12
      16 12 16 13 19
      17 15 20 12 16
      19 16 17 15 20
      12 13 20 20 10
      20 19 15 18 19
      20 13 10 11 14
      19 19 13 14 15
      15 13 17 17 20
      14 18 14 11 20

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

[51]: 10-element Vector{Int64}:
      14
      12
      15
      16
      13
      19
      13
      19
      13
      18

```

Рисунок 4.7: Примеры использования массивов

```

15 18 19 19 20
10 11 13 14 20
13 14 15 19 19
13 15 17 17 20
11 14 14 18 20

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

[58]: 10x5 BitMatrix:
0 0 0 0 0
1 0 1 0 1
1 1 1 0 1
1 1 1 1 1
0 0 1 1 0
1 1 1 1 1
1 0 0 0 0
1 1 0 0 1
1 0 1 1 1
0 1 0 0 1

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

[59]: 29-element Vector{CartesianIndex{2}}:
 CartesianIndex(2, 1)
 CartesianIndex(3, 1)
 CartesianIndex(4, 1)
 CartesianIndex(6, 1)
 CartesianIndex(7, 1)
 CartesianIndex(8, 1)
 CartesianIndex(9, 1)
 CartesianIndex(3, 2)
 CartesianIndex(4, 2)
 CartesianIndex(6, 2)
 CartesianIndex(8, 2)
 CartesianIndex(10, 2)
 CartesianIndex(2, 3)
      :
 CartesianIndex(9, 3)
 CartesianIndex(4, 4)
 CartesianIndex(5, 4)
 CartesianIndex(6, 4)
 CartesianIndex(9, 4)
 CartesianIndex(2, 5)
 CartesianIndex(3, 5)
 CartesianIndex(4, 5)
 CartesianIndex(6, 5)
 CartesianIndex(8, 5)
 CartesianIndex(9, 5)
 CartesianIndex(10, 5)

```

Рисунок 4.8: Примеры использования массивов

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 $P = A \cap B \cup A \cap B \cup A \cap C \cup B \cap C$ (рис. 4.9).

```
[60]: #Задание 1
      A = Set([0,3,4,9])
      B = Set([1,3,4,7])
      C = Set([0,1,2,4,7,8,9])

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

[60]: Set{Int64} with 6 elements:
      0
      4
      7
      9
      3
      1
```

Рисунок 4.9: Задание №1. Работа с множествами

Задание №2

Приведем свои примеры с выполнением операций над множествами элементов разных типов (рис. 4.10).

```

[61]: #Задание 2
A = Set(["Иванов", "Петров", "Сидоров"])
B = Set(["Иванов", "Пушкин"])#объединение
print(union(A,B))
Set(["Иванов", "Петров", "Пушкин", "Сидоров"])

[62]: Set1 = Set([1, 2, 3, "Hello"])

[62]: Set{Any} with 4 elements:
      2
      "Hello"
      3
      1

[63]: println("\nElements of set:")
      for i in Set1
          println(i)
      end

      Elements of set:
      2
      Hello
      3
      1

[64]: print(in("Hello", Set1))

      true

[66]: Set1 = push!(Set1, "World")
      println("\nSet after adding one element: \n", Set1)

      Set after adding one element:
      Set{Any[2, "Hello", "World", 3, 1]}

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

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

```

Рисунок 4.10: Задание №2. Примеры операций над множествами элементов разных типов

Задание №3

Создадим массивы разными способами, используя циклы (рис. 4.11 - рис. 4.17).

[illegible]

Рисунок 4.11: Задание №3. Работа с массивами

```
[86]: S8 = vcat(fill(1:2*tmp[1], 1), fill(2*tmp[2], 1), fill(2*tmp[3], 4))
count_6 = count(x -> x == 6, S8)
print(S8, '\n', count_6)

[16, 64, 0, 0, 0, 8]
0

[87]: # 3.10
using Statistics
f(x) = exp(x)*cos(x) #здесь у нас f так как такая функция уже создавалась
Y = [f(x) for x in 3:0.1:6]
mean(Y)

[87]: 53.11374594642971

[88]: # 3.11
x_vals = [0.1^i for i in 3:36]
y_vals = [0.2^j for j in 1:334]
arr11 = vcat(x_vals, y_vals)

[88]: 24-element Vector{Float64}:
 0.00100000000000000002
 1.00000000000000004e-6
 1.00000000000000005e-9
 1.00000000000000006e-12
 1.00000000000000009e-15
 1.0000000000000001e-18
 1.000000000000012e-21
 1.000000000000014e-24
 1.000000000000015e-27
 1.000000000000017e-30
 1.000000000000018e-33
 1.00000000000002e-36
 0.2
 0.0010000000000000003
 1.2800000000000005e-5
 1.0724000000000006e-7
 8.192000000000005e-10
 6.553600000000005e-12
 5.242800000000005e-14
 4.194304000000005e-16
 3.3554432000000048e-18
 2.684354560000004e-20
 2.1474836480000035e-22
 1.717986918400003e-24
```

Рисунок 4.12: Задание №3. Работа с массивами

```
1.7179809218400000e-24

# 2 12
M = 25
arr12 = ["1" / i for i in 1:M]

25-element Vector{Float64}:
 2.0
 2.0
 2.0666666666666665
 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
27594.105763157893
52420.0
99864.38095238095
190650.18181818182
364722.808956217
699050.6666666666
1.34217728e6

# 3 13
N = 30
arr13 = ["fns1" for i in 1:N]

30-element Vector{String}:
"fns1"
"fns2"
"fns3"
"fns4"
"fns5"
"fns6"
"fns7"
"fns8"
"fns9"
"fns10"
"fns11"
"fns12"
"fns13"
⋮
"fns19"
"fns20"
"fns21"
"fns22"
"fns23"
⋮
"fns29"
"fns30"
```

Рисунок 4.13: Задание №3. Работа с массивами

```
(131) # Cosinus (pi/2) - 0,0000000000000000
diff1 = [y[i+1] - x[i] for i in 1:n-1]
print(diff1)
[134, -236, 159, -423, -440, -417, -236, 293, 21, 765, 240, 765, 65, 200, 483, 158, -78, 8, -122, -100, 75, -218, 418, 14, -249, -389, -422, 267, 181, 742, -430, -58, 186, 773, 479, 275, -385, 134, -441,
31, 902, 438, -389, -181, 49, 456, 137, 80, 13, 228, -468, 734, 542, 342, -122, 188, 385, 536, 866, 503, 73, -208, -501, 84, 484, -244, 212, -586, -291, -821, -283, -247, 178, 533, 8, 944, -576, -141,
211, 118, 652, 387, 3, -118, 802, 526, -386, -182, 335, 170, 109, 440, 66, 526, 917, 438, -281, -446, 711, -883, 556, -566, 120, -259, -286, -84, -436, -445, -529, -306, 533, -331, -57, 307, 181, 185, -14,
12, -41, 307, 95, -462, 571, 117, 386, -51, 52, 228, -354, 571, 263, 101, 384, 834, 459, 25, -136, 21, 235, 159, 16, 316, -477, 451, 189, 496, 18, -265, 151, 589, -136, 916, -789, -742, -200, 633, 170, 10,
1, 8, 175, -179, 78, -552, -468, -144, 175, -54, -664, -134, -486, 85, 86, 129, 130, 508, 9, 123, 320, 541, 181, 205, 189, 866, 125, 964, -256, -150, 307, 295, -34, -546, 380, 108, 644, 151, 10, -223,
710, -680, 114, 39, -309, 44, -29, -749, 223, 31, 152, 49, -369, -111, -46, 13, 639, 239, 238, 2, 124, -484, 18, -60, 539, -937, 385, 238, -876, 434, -22, 658, -402, -88, 122, -383, 93, -249, -865, -232,
3, -758, -435, 222, -282, 34, -238, -279, 260, -473, -118, 413]

(141) # Cosinus (pi/2) - 0,0000000000000000
diff2 = [y[i] - 2x[i+1] - x[i+1] for i in 1:n-1]
print(diff2)
[1396, 1765, 1271, 803, 1753, 2089, 9, 1171, 2154, 1525, 596, 427, 1235, 1556, -16, 87, 1402, 2553, 1183, 135, 1037, 1161, -272, 713, 489, 1529, 1353, 1164, -169, 530, 9579, 826, -45, 544, -145, 946, 1536,
697, 866, -58, 145, 1588, 1101, 439, 296, 899, 1322, 1644, -49, 1177, 719, 182, 206, 1777, 2117, 986, -143, -21, 952, 1861, 1842, 1451, 1208, 366, 1677, 987, 1588, 1581, 1825, 1674, 1395, 889, 59, 8, 1608,
949, 1053, 413, 723, 889, 123, -382, 518, 124, 181, -1097, -955, 1052, 726, 1278, 839, 984, 101, 516, 617, 879, 2189, 318, 718, 773, 1177, 866, 1014, 1131, 688, 1228, 1554, 1551, 2118, 595, 625, 724, 1828,
177, 588, 1778, 1817, 1498, 1348, 480, 1868, 487, 2025, 1809, 612, 881, 111, 1331, 622, 589, 1169, 16, 1550, 1259, 753, 683, 2182, 787, 134, 1868, 1149, 1474, 423, 1368, -52, 1180, 1830, 982, 223, 676, 9,
6, 1119, 1863, -2096, 31, -411, -2141, 943, -225, 947, 1777, 539, 1486, 1528, 1640, 26, 1369, 1317, -240, 1572, 269, 1133, 541, -388, -353, 1147, 649, 937, -73, 1276, 788, -252, 1693, 886, 481, 725, 1256, 986,
9, 987, -429, 1559, 646, 987, 128, 953, 2825, 2273, 885, 489, 747, 154, 953, 1867, 1574, 1734, 479, -189, 879, 1484, 1138, 2053, 389, 612, 1681, 141, 273, 1526, 2142, 1577, 486, 2132, 853, 432, 2867, 774,
46, 78, 1106, 1050, 1267, 1545, 871, 668, 1776, 2832, 499, 1534, 2871, 1358, 1729, 222, 946, 1627, 759, 1694, 4593, 739]

(151) # Cosinus (pi/2) - 0,0000000000000000
diff3 = [sin(y[i]) / cos(x[i+1]) for i in 1:n-1]
print(diff3)
```

Рисунок 4.14: Задание №3. Работа с массивами

```

[97]: # Recursive count
def xk = comb[exp->[i+1]] / (x[i] + 10) for i in [n-1]
return xk
0.00911031757055483

[98]: # Recursive y > 600
y.big = y[y > 600]
println("maxima y > 600: by big")
indices = findall(y > 600)

maxima y > 600: [672, 676, 680, 684, 688, 692, 696, 700, 704, 708, 712, 716, 720, 724, 728, 732, 736, 740, 744, 748, 752, 756, 760, 764, 768, 772, 776, 780, 784, 788, 792, 796, 800, 804, 808, 812, 816, 820, 824, 828, 832, 836, 840, 844, 848, 852, 856, 860, 864, 868, 872, 876, 880, 884, 888, 892, 896, 900, 904, 908, 912, 916, 920, 924, 928, 932, 936, 940, 944, 948, 952, 956, 960, 964, 968, 972, 976, 980, 984, 988, 992, 996]

[99]: 102-element Vector{Int64}:
 1
 4
10
12
13
15
16
18
20
22
23
24
26
28
29
31
33
35
219
221
223
225
226
228
229
230
233
235
242
247

[100]: # Recursive y < 600
y.big = y[y < 600]
println("maxima y < 600: by big")
indices = findall(y < 600)

maxima y < 600: [672, 676, 680, 684, 688, 692, 696, 700, 704, 708, 712, 716, 720, 724, 728, 732, 736, 740, 744, 748, 752, 756, 760, 764, 768, 772, 776, 780, 784, 788, 792, 796, 800, 804, 808, 812, 816, 820, 824, 828, 832, 836, 840, 844, 848, 852, 856, 860, 864, 868, 872, 876, 880, 884, 888, 892, 896, 900, 904, 908, 912, 916, 920, 924, 928, 932, 936, 940, 944, 948, 952, 956, 960, 964, 968, 972, 976, 980, 984, 988, 992, 996]

[101]: 102-element Vector{Int64}:
 1
 4
10
12
13
15
16
18
20
22
23
24
26
28
29
31
33
35
219
221
223
225
226
228
229
230
233
235
242
247

```

Рисунок 4.15: Задание №3. Работа с массивами

```

UnderError: 'vect' y not defined in 'Main'
Suggestion: check for spelling errors or missing imports.

Stacktrace:
 [1] top-level scope

[110]: # Recursive y < 600
y.big = y[y < 600]
count_near_max = count(y < 600)

[111]: 50

[112]: # Recursive y < 600
even_count = count(iseven, y)
odd_count = count(isodd, y)

[113]: 112

[114]: # Recursive y < 600
even_count = count(iseven, y)

[115]: 118

[116]: # Recursive y < 600
multiple_7 = count(x >= 7 & 7 == 0, x)

[117]: 39

[118]: # Recursive y < 600
sorted_x_by_y = x[sortperm(y)]

[119]: 250-element Vector{Int64}:
 806
 655
 317
 244
 548
 478
 220
 653
 787
 283
 529
 178
 642
 1
179
184
967
681
873
78
481
512
484
457
418
820

```

Рисунок 4.16: Задание №3. Работа с массивами

над ними для решения задач.