Отчёт по лабораторной работе №2

Компьютерный практикум по статистическому анализу данных

Канева Екатерина, НФИбд-02-22

Содержание

Список иллюстраций

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

- Используя Jupyter Lab, повторить примеры.
- Выполнить задания для самостоятельной работы.

3 Теоретическая часть

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.

Для выполнения заданий была использована официальная документация Julia.

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

Сначала я выполнила примеры с кортежами (рис. 4.1):

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

[5]: ("Python", "Julia", "R")

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

[7]: (1, 2, 3)

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

[9]: (1, 2.0, "tmp")

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

[11]: (a = 2, b = 3)

[13]: # длина кортежа x2:
      length(x2)

[13]: 3

[15]: x2[1], x2[2], x2[3]

[15]: (1, 2.0, "tmp")

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

[17]: 5

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

[19]: (2, 3, 3)

[21]: # проверка вхождения элементов tmp и 0 б кортеж x2
      # (два способа обращения к методу in()):
      in("tmp", x2), 0 in x2

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

Рис. 4.1: Примеры с кортежами.

Потом я выполнила примеры со словарями (рис. 4.2):

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

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

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

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

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

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

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

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

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

[31]: true

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

[47]: Dict{String, Any} with 3 entries:
        "Сидоров П.С." => "555-3344"
        "Бухгалтерия"  => "555-2368"
        "Иванов И.И."  => ("867-5309", "333-5544")

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

[49]: Dict{String, Any} with 2 entries:
        "Сидоров П.С." => "555-3344"
        "Бухгалтерия"  => "555-2368"

[53]: # 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)

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

Рис. 4.2: Примеры со словарями.

Потом я выполнила примеры со множествами (рис. 4.3):

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

       Set([5, 4, 3, 1])

[317]: # создать множество из 11 символьных значений:
B = Set("abrakadabra")
print(B)

       Set(['a', 'd', 'r', 'k', 'b'])

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

[59]: false

[327]: S3 = Set([1,2,2,3,1,2,3,2,1]);
S4 = Set([2,3,1]);
issetequal(S3,S4)

[327]: true

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

       Set([4, 2, 3, 1])

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

       Set([2, 1])

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

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

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

[335]: true

[337]: # добавление элемента в множество:
push!(S4, 99)
print(S4)

       Set([2, 99, 3, 1])

[339]: # удаление последнего элемента множества:
pop!(S4)
print(S4)

       Set([99, 3, 1])
```

Рис. 4.3: Примеры со множествами.

Потом я выполнила примеры с массивами (рис. 4.4-4.9):

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

[81]: Any[]

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

[83]: Float64[]

[341]: # вектор-столбец:
       a = [1, 2, 3]
       print(a)

       [1, 2, 3]

[343]: # вектор-строка:
       b = [1 2 3]
       print(b)

       [1 2 3]

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

[89]: ([1 4 7; 2 5 8; 3 6 9], [1 2 3; 4 5 6; 7 8 9])

[91]: A

[91]: 3×3 Matrix{Int64}:
       1  4  7
       2  5  8
       3  6  9

[93]: B

[93]: 3×3 Matrix{Int64}:
       1  2  3
       4  5  6
       7  8  9

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

[97]: 1×8 Matrix{Float64}:
       0.033723  0.123527  0.910441  0.0505996  …  0.885282  0.273298  0.807926

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

[101]: 2×3 Matrix{Float64}:
        0.213299  0.900926  0.787135
        0.763103  0.405134  0.268585
```

Рис. 4.4: Примеры с массивами 1.

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

[103]: 4×3×2 Array{Float64, 3}:
       [:, :, 1] =
        0.235311  0.43711   0.0171069
        0.367035  0.852674  0.285454
        0.61061   0.443039  0.868912
        0.792952  0.30676   0.813905

       [:, :, 2] =
        0.305917  0.833856  0.547395
        0.298329  0.159213  0.97397
        0.897778  0.158119  0.109487
        0.339145  0.660809  0.359739

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

       [1.0, 1.4142135623730951, 1.7320508075688772, 2.0, 2.23606797749979, 2.449489742783178, 2.6457513110645907, 2.8284271247461903, 3.0, 3.1622776601683795]

[347]: # 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]
       print(ar_1)

       [3, 27, 75, 147, 243]

[349]: # 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)]
       print(ar_2)

       [1, 9, 49, 81]

[351]: # одномерный массив из пяти единиц:
       print(ones(5))

       [1.0, 1.0, 1.0, 1.0, 1.0]

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

[113]: 2×3 Matrix{Float64}:
        1.0  1.0  1.0
        1.0  1.0  1.0

[353]: # одномерный массив из 4 нулей:
       print(zeros(4))

       [0.0, 0.0, 0.0, 0.0]

[117]: # заполнить массив 3x2 цифрами 3.5
       fill(3.5,(3,2))

[117]: 3×2 Matrix{Float64}:
        3.5  3.5
        3.5  3.5
        3.5  3.5
```

Рис. 4.5: Примеры с массивами 2.

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

```
[121]: 3×6 Matrix{Int64}:
 1  2  1  2  1  2
 1  2  1  2  1  2
 1  2  1  2  1  2
```

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

```
[123]: 2×6 Matrix{Int64}:
 1  3  5  7   9  11
 2  4  6  8  10  12
```

```
[125]: # транспонирование
b'
```

```
[125]: 6×2 adjoint(::Matrix{Int64}) with eltype Int64:
  1   2
  3   4
  5   6
  7   8
  9  10
 11  12
```

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

```
[127]: 6×2 transpose(::Matrix{Int64}) with eltype Int64:
  1   2
  3   4
  5   6
  7   8
  9  10
 11  12
```

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

```
[129]: 10×5 Matrix{Int64}:
 16  19  13  10  19
 14  10  11  18  19
 20  13  10  15  20
 17  18  12  17  16
 14  12  15  10  12
 11  19  14  15  16
 12  13  14  15  11
 17  10  19  13  18
 11  12  20  20  19
 18  19  19  19  15
```

Рис. 4.6: Примеры с массивами 3.

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

```
[131]: 10-element Vector{Int64}:
 19
 10
 13
 18
 12
 19
 13
 10
 12
 19
```

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

```
[133]: 10×2 Matrix{Int64}:
 19  19
 10  19
 13  20
 18  16
 12  12
 19  16
 13  11
 10  18
 12  19
 19  15
```

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

```
[135]: 10×3 Matrix{Int64}:
 19  13  10
 10  11  18
 13  10  15
 18  12  17
 12  15  10
 19  14  15
 13  14  15
 10  19  13
 12  20  20
 19  19  19
```

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

```
[137]: 3×2 Matrix{Int64}:
 14  19
 17  16
 11  16
```

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

```
[139]: 3-element Vector{Int64}:
 13
 10
 19
```

Рис. 4.7: Примеры с массивами 4.

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

[141]: 10×5 Matrix{Int64}:
        11  10  10  10  11
        11  10  11  10  12
        12  12  12  13  15
        14  12  13  15  16
        14  13  14  15  16
        16  13  14  15  18
        17  18  15  17  19
        17  19  19  18  19
        18  19  19  19  19
        20  19  20  20  20

[143]: # сортировка по строкам:
       sort(ar,dims=2)

[143]: 10×5 Matrix{Int64}:
        10  13  16  19  19
        10  11  14  18  19
        10  13  15  20  20
        12  16  17  17  18
        10  12  12  14  15
        11  14  15  16  19
        11  12  13  14  15
        10  13  17  18  19
        11  12  19  20  20
        15  18  19  19  19

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

[145]: 10×5 BitMatrix:
        1  1  0  0  1
        0  0  0  1  1
        1  0  0  1  1
        1  1  0  1  1
        0  0  1  0  0
        0  1  0  1  1
        0  0  0  1  0
        1  0  1  0  1
        0  0  1  1  1
        1  1  1  1  1
```

Рис. 4.8: Примеры с массивами 5.

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

[147]: 28-element Vector{CartesianIndex{2}}:
        CartesianIndex(1, 1)
        CartesianIndex(3, 1)
        CartesianIndex(4, 1)
        CartesianIndex(8, 1)
        CartesianIndex(10, 1)
        CartesianIndex(1, 2)
        CartesianIndex(4, 2)
        CartesianIndex(6, 2)
        CartesianIndex(10, 2)
        CartesianIndex(5, 3)
        CartesianIndex(8, 3)
        CartesianIndex(9, 3)
        CartesianIndex(10, 3)
        ⋮
        CartesianIndex(6, 4)
        CartesianIndex(7, 4)
        CartesianIndex(9, 4)
        CartesianIndex(10, 4)
        CartesianIndex(1, 5)
        CartesianIndex(2, 5)
        CartesianIndex(3, 5)
        CartesianIndex(4, 5)
        CartesianIndex(6, 5)
        CartesianIndex(8, 5)
        CartesianIndex(9, 5)
        CartesianIndex(10, 5)
```

Рис. 4.9: Примеры с массивами 6.

Далее я приступила к выполнению заданий 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 $P = A \cap B \cup A \cap B \cup A \cap C \cup B \cap C$ (рис. 4.10):

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

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

[176]: union(intersect(A, B), intersect(A, C), intersect(B, C))

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

Рис. 4.10: Задание 1.

2. Приведём примеры с выполнением операций над множествами элементов разных типов (рис. 4.11):

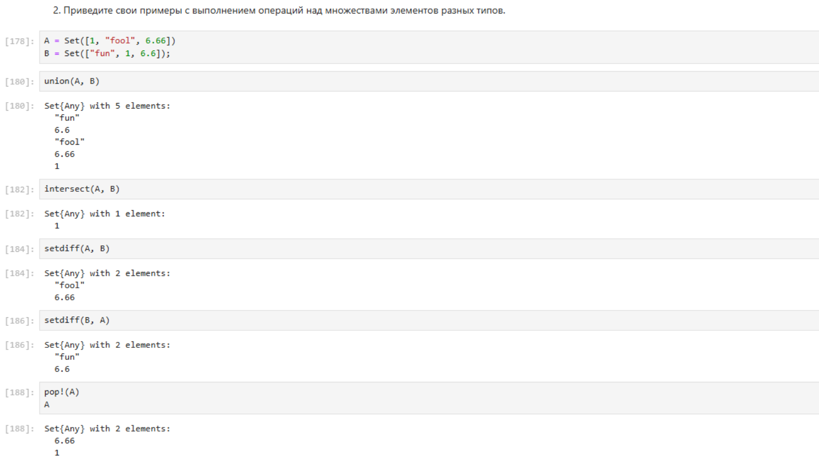

Рис. 4.11: Задание 2.

3. Создадим массивы разными способами (рис. 4.12-4.17):

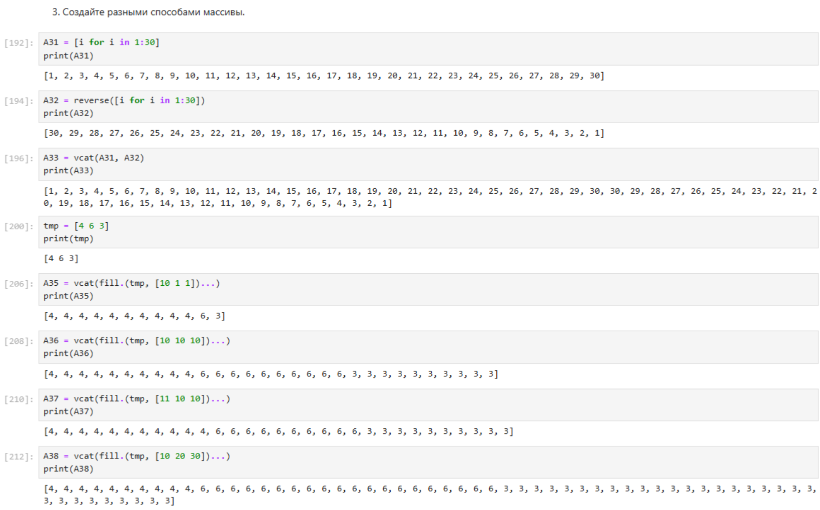

Рис. 4.12: Задание 3 (1-8).

[228]:
```
tmp1 = []

for i in 1:3
    push!(tmp1, 2^tmp[i])
end

A39 = vcat(fill.(tmp1, [1, 1, 4])...)

count = 0
for i in A39
    if '6' in string(i)
        count += 1
    end
end

print(A39, '\n', count)
```

```
[16, 64, 8, 8, 8, 8]
2
```

[232]:
```
f(x) = exp(x) * cos(x)

y = [f(x) for x in 3:0.1:6]

print(sum(y)/length(y))
```

```
53.11374594642971
```

[234]:
```
A311 = []

for j in 1:3:34
    i = j + 2
    push!(A311, ((0.1)^i, (0.2)^j))
end

print(A311)
```

```
Any[(0.0010000000000000002, 0.2), (1.0000000000000004e-6, 0.0016000000000000003), (1.0000000000000005e-9, 1.2800000000000005e-5), (1.0000000000000006e-12, 1.0240000000000006e-7), (1.0000000000000009e-15, 8.192000000000005e-10), (1.000000000000001e-18, 6.553600000000055e-12), (1.0000000000000012e-21, 5.242880000000056e-14), (1.0000000000000014e-24, 4.194304000000005e-16), (1.0000000000000015e-27, 3.3554432000000048e-18), (1.0000000000000017e-30, 2.684354560000004e-20), (1.0000000000000018e-33, 2.1474836480000035e-22), (1.000000000000002e-36, 1.717986918400003e-24)]
```

[236]:
```
A312 = []

for i in 1:25
    push!(A312, (2^i)/i)
end

print(A312)
```

```
Any[2.0, 2.0, 2.6666666666666665, 4.0, 6.4, 10.666666666666666, 18.285714285714285, 32.0, 56.888888888888886, 102.4, 186.1818181818182, 341.33333333333, 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]
```

[238]:
```
A313 = []

for i in 1:30
    push!(A313, "fn$i")
end

print(A313)
```

```
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"]
```

Рис. 4.13: Задание 3 (9-13).

[242]:
```
x = []
y = []

for i in 1:250
    push!(x, rand(0:999))
    push!(y, rand(0:999))
end
```

[258]:
```
print(x)
```

```
Any[81, 15, 343, 4, 300, 642, 833, 286, 73, 7, 848, 49, 549, 700, 115, 418, 839, 887, 757, 745, 974, 506, 423, 811, 313, 154, 18, 264, 637, 153, 597, 84
4, 87, 588, 353, 557, 796, 785, 464, 462, 879, 988, 387, 473, 736, 102, 95, 188, 301, 917, 141, 262, 870, 249, 566, 862, 794, 949, 835, 390, 306, 228, 74
5, 322, 217, 883, 691, 786, 772, 159, 954, 942, 771, 436, 10, 334, 303, 611, 532, 950, 884, 788, 154, 730, 547, 356, 432, 595, 382, 484, 720, 459, 367,
3, 957, 864, 592, 35, 483, 897, 477, 507, 247, 539, 950, 422, 689, 864, 461, 106, 52, 10, 641, 377, 305, 772, 402, 642, 433, 414, 915, 611, 915, 86, 55,
344, 652, 935, 964, 381, 527, 256, 857, 149, 803, 255, 34, 575, 239, 170, 343, 967, 681, 266, 464, 55, 949, 278, 202, 240, 641, 639, 574, 975, 849, 429,
137, 610, 83, 83, 592, 830, 336, 841, 703, 5, 42, 90, 388, 339, 638, 75, 555, 656, 112, 896, 560, 787, 469, 931, 650, 743, 365, 612, 376, 821, 480, 330,
125, 898, 283, 334, 284, 92, 476, 730, 174, 648, 866, 388, 267, 75, 898, 764, 554, 800, 864, 746, 184, 576, 147, 659, 773, 483, 59, 823, 990, 936, 451, 9
03, 505, 870, 877, 759, 985, 462, 803, 235, 264, 556, 715, 617, 693, 904, 539, 518, 31, 658, 414, 847, 593, 340, 284, 208, 388, 818, 933, 786, 512, 811]
```

[256]:
```
print(y)
```

```
Any[947, 585, 848, 310, 502, 480, 105, 878, 694, 324, 656, 326, 787, 805, 327, 187, 260, 360, 221, 198, 649, 793, 674, 919, 492, 96, 29, 787, 489, 116, 2
2, 255, 894, 160, 305, 632, 73, 671, 578, 753, 507, 568, 744, 581, 288, 231, 697, 946, 596, 974, 213, 163, 98, 306, 585, 831, 771, 868, 718, 469, 437, 22
6, 705, 906, 234, 856, 610, 709, 866, 947, 763, 517, 536, 970, 135, 240, 699, 328, 284, 300, 906, 26, 118, 434, 372, 967, 517, 372, 584, 520, 141, 902, 1
72, 694, 875, 550, 808, 986, 947, 286, 409, 956, 519, 169, 102, 570, 674, 939, 656, 975, 968, 521, 232, 183, 65, 631, 291, 769, 103, 193, 995, 951, 285,
228, 711, 426, 759, 604, 688, 266, 1, 516, 843, 870, 22, 207, 804, 205, 895, 663, 304, 490, 258, 926, 333, 432, 895, 852, 971, 500, 590, 894, 773, 971, 2
82, 784, 928, 569, 859, 257, 945, 29, 578, 972, 396, 242, 407, 693, 424, 844, 487, 232, 419, 40, 828, 646, 540, 927, 348, 23, 846, 146, 119, 28, 999, 62
2, 388, 805, 532, 142, 426, 171, 88, 9, 425, 863, 637, 786, 532, 323, 361, 633, 249, 914, 695, 999, 918, 472, 496, 530, 639, 469, 269, 180, 479, 483, 89
9, 396, 425, 397, 808, 148, 626, 972, 562, 947, 987, 183, 520, 657, 314, 911, 883, 178, 171, 57, 926, 510, 279, 808, 194, 374, 417, 715, 193, 794, 739, 6
2, 343, 444]
```

[244]:
```
b1 = []

for i in 1:249
    push!(b1, y[i+1] - x[i])
end

print(b1)
```

```
Any[504, 833, -33, 498, 180, -537, 45, 408, 251, 649, -522, 738, 256, -373, 72, -158, -479, -666, -559, -96, -181, 168, 496, -319, -217, -125, 769, 225,
-521, -131, -342, 50, 73, -283, 279, -484, -125, -207, 289, 45, -311, -244, 194, -185, -505, 595, 851, 408, 673, -704, 22, -164, -564, 336, 265, -91, 74,
-231, -366, 47, -80, 477, 161, -88, 639, -273, 18, 80, 175, 604, -437, -406, 199, -301, 230, 365, 25, -327, -232, -44, -858, -670, 280, -358, 420, 161, -
60, -11, 138, -343, 182, -287, 327, 872, -407, -56, 394, 912, -197, -488, 479, 12, -78, -437, -380, 252, 250, -208, 514, 862, 469, 222, -458, -312, 326,
-481, 367, -539, -240, 581, 36, -326, -687, 625, 371, 415, -48, -247, -698, -380, -11, 587, 13, -127, -596, 549, 171, 320, 424, 134, 147, -709, 245, 67,
-32, 840, -97, 693, 298, 350, 253, 134, 397, -693, -65, 499, 432, 249, 174, 862, -563, -252, 636, -445, -461, 402, 651, 334, 456, 148, -406, 344, -515, 1
72, 534, -356, 367, -439, -446, -85, -504, -624, -337, 387, 246, -433, 325, 202, 17, -472, -112, -246, -275, 333, 387, -93, 612, -116, -543, -27, 366, 17
4, 16, -69, 445, 118, -392, -250, 346, 63, 322, -390, -593, -4, 424, 76, -594, -511, -54, -95, -357, -244, 95, -197, -38, 525, -620, 285, 393, -242, 196,
266, -515, -733, -482, 408, 479, -379, 394, -653, -219, 77, 431, -15, 406, -79, -871, -443, -68]
```

[260]:
```
b2 = []

for i in 1:248
    push!(b2, x[i] + 2*x[i+1] - x[i+2])
end

print(b2)
```

```
Any[-232, 697, 51, -38, 751, 2022, 1332, 425, -761, 1654, 397, 447, 1834, 512, 112, 1209, 1856, 1656, 1273, 2187, 1563, 541, 1732, 1283, 603, -74, -91, 1
385, 346, 503, 2190, 430, 910, 737, 671, 1364, 1902, 1251, 509, 1232, 2468, 1289, 597, 1843, 845, 104, 170, -127, 1994, 937, -205, 1753, 802, 519, 1496,
1501, 1857, 2229, 1309, 774, 17, 1396, 1172, -127, 1292, 1479, 1491, 2171, 136, 1125, 2067, 2048, 1633, 122, 375, 329, 993, 725, 1548, 1930, 2306, 366, 1
067, 1468, 827, 625, 1240, 875, 630, 1465, 1271, 1190, -584, 1053, 2093, 2013, 179, 104, 1800, 1344, 1244, 462, 375, 2017, 1105, 936, 1956, 1680, 621, 20
0, -569, 915, 1090, 215, 1447, 934, 1253, 1094, 346, 1633, 1222, 2355, 1032, -148, 91, 713, 1558, 2482, 1199, 1179, 182, 1821, 352, 1500, 1279, -252, 94
5, 883, 236, -111, 1596, 2063, 749, 1139, -375, 1675, 1303, 442, 41, 883, 1345, 812, 1675, 2244, 1570, 93, 1274, 693, -343, 437, 1916, 661, 1315, 2242, 6
71, -1, -166, 527, 428, 1540, 233, 529, 1755, -16, 1344, 1229, 1665, 794, 1681, 1488, 1771, 861, 1213, 543, 1538, 1451, 1015, -318, 1638, 1130, 667, 810,
-8, 314, 1762, 430, 604, 1992, 1375, 847, -481, 1107, 1872, 1072, 1290, 1782, 2172, 538, 1189, 211, 692, 1722, 1680, -222, 715, 1867, 2411, 935, 1752, 10
43, 1368, 1865, 1410, 2267, 1106, 1833, 1009, 207, 661, 1369, 1256, 1099, 1962, 1464, 1544, -78, 933, 639, 1515, 1693, 989, 700, 312, 166, 1091, 1898, 19
93, 999]
```

Рис. 4.14: Задание 3 (14.1-2).

[264]:
```
b3 = []

for i in 1:249
    push!(b3, sin(y[i])/cos(x[i+1]))
end

print(b3)
```

```
Any[1.2925956198062198, -0.7301023493554736, 0.34880407576324574, -38.50759602292147, -1.3838570599994096, -0.693758470677355, 0.9769962386470165, 1.3545064492838874, 0.3818075598421599, -0.41499509455536883, 1.8587921139366268, 0.9322778740635587, -1.1911728476319317, -2.0972534397874174, -0.2748119801611515, 1.0163869814066446, 1.4253182501168 8, -0.9663148762283285, -0.9803825933891249, -0.08003011190537527, -0.9864743810964696, -2.2000120552295996, 1.11187895184277 07, 2.4910733374598803, -0.9443164346300404, 1.4895696764207433, -0.6673951460282583, -1.3576104103852094, 1.4980660282727823, 0.23778832563795615, 0.019080938604351408, -0.8887956785097432, -1.126721699592327, 0.5273131812083632, 0.4437588255368719, 1.3398666960647194, -0.7342054363951513, -1.6702505167550512, 0.05395277319795409, -1.041587282224608, -30.21991798877871, -0.7045570424662444, -2.7967554697586827, 0.2989986260126441 5, -8.42150361339 9432, -1.3636267360893493, -0.4775615884151746, -0.4479393643118581 6, -0.8341079436703143, -0.11385090598393548, 1.8518286230754475, 0.3637707063141857 3, 0.8352679013614326, -1.0948711155299178, 1.7161797002604195, -1.4686953854608, -0.9944034383686319, 1.0102937906485 28, 1.0948015249132965, 2.611755243781 4378, 1.3479906100951073, 0.2140655219023309, 72.39147969314365, -0.9649857783841111, -1.0217924828471394, 1.0078491692773 92, 0.6143629123766351, -1.2492 558143428618, 2.5745098485342655, -1.9536914212678034, 0.4457940421427337, -3.7917814056526584, -1.2056583416164237, -0.8142281753556408, 0.1613438640684 0706, 5.801663747303487, 25.159261308989897, -1.9950207553852985, 2.9198434315745362, 2.8506409081643898, -1.0952592394741 94, -0.7640245411303707, -2.4065 861303964047, 0.47507717475802297, -1.7794684815180697, -18.49072055125118, -3.00363990545109, 3.2905964290179 84, -0.3362898967652507, 1.1892124219787 488, 0.3835540751292311, 0.4206745877343656, -0.7158227386730719, -0.7663754779958716, -0.9997153711219673, -1.1608601181176008, 0.6345426701108157, -0.640 8145343211716, -13.053542258424917, -0.13246092144359495, -1.5565150589887078, -2.1750017114074254, -2.7651308023963663, -1.8481883541506998, 1.921464606 570601, 1.7898292772215127, -0.993681629014393, -0.4805555524015433 5, 0.8139111379248708, -5.485200919024605, -0.4525137467312882, -0.4864840154210756, -0.4598948611886767, -0.7344157164277201, 1.2275652312569867, 0.4474867603493332, 2.090519104426 17, 0.7422216406175 02, 0.8079225816611907, 1.3994221244322 558, 19.462297800907386, -1.1226257192572238, -2.0168744502203966, 43.95687515019683 6, -191.44425735856 02, -7.975892179406191, -2.5960758981806404, -0.81 5186766894655, -0.01158953892757652 4, 1.2184516415 0134, -21.147396688671737, -0.8855938371941715, -3.8839362634888928, 0.6909486630474903, 0.0102647248 7186644, 0.3986765004546626, 0.2461578595830580 8, -0.7358053761927487, 0.3694739523444127 7, 0.1465126965514 5127, 0.8179836531648 41, 0.1180844143415 9574, -0.7440874484235722, 1.2062094898618747, -0.3986651278413003 7, -1.0287662414351941, 11.19940952687148, -0.9940785954591237, -0.7530808289899744, -0.47088 745207285926, 1.8790522061718555, -1.5941527247461766, 0.3737005384797176, -0.3417357452966212, 3.9403868097466126, -2.948048952006372, -1.0930736550029 931, -1.4555877427622208, -3.9058334318807115, -3.032509171212003, 0.7130632320215894, 0.6712124529429808, -0.0907534108999655 3, -1.2578036735084972, 0.55 9300603207385 1, 0.2430386913605951, 2.2019209407235145, 72.246857033400 12, 0.1198273823726621 6, -0.9159062598460014, -0.0576225829228374 8, 0.943890617780 1788, 1.1093567717678727, 1.6341308067077 8, 1.2298214296471466, -1.3157784214317865, 11.167702573899 85, 0.3682670411569162, 1.4166375056209304, 0.8884844 0304521 14, 59.26882521470606, 1.1876087018015 21, 0.4533061888223646, 0.4945752120563753 4, 0.0526590322630793 6, 0.0448604388431658 9, 1.0087900000358871, 0.8674535306630503, -0.9975818131146514, -0.6077015246019205, -1.736421988678864, 3.1605490111814696, -0.0565066908194980 5, 8.463483663119904, -1.8967120 626433251, -2.2988481483424383, 1.0048480416201397, 1.2002348679839558, -65.951120856233 1, -0.5517467340460189, 0.3031040623822514 6, -1.1361670202828 6, 0.8768649621193 49, 0.4284598958150171 4, 1.4512245296160433, 0.0265117252429356 16, -4.7622923352743545, -3.2059902212562 28, 0.7830153013850608, -1.0100614 907283951, -1.2814338431284 88, -0.7962367587243709, -1.3326937254696987, 1.0390003683947775, 1.0004192718461316, 0.7822546952042169, 0.4926708728933346, 0.8791551842713702, 3.7496764985560 69, -1.3106961792420135, 0.5877472960982096, 0.3846184035497 2715, -2.4388199230797394, -8.583775636034542, -0.3442059 4816584604, -3.0929542265546 89, -0.6313645199909307, 0.7126960791560 416, 0.9997089761430 94, -1.3956736055939039, -0.4997074120327 4166, 0.2246558595150294 2, -0.2962322681748648, 4.0786314401569 59, -1.0445815273735812, 0.4768170529400889, -4.2718241795818 7, 1.1325733733540517, 1.6937318324029709, 0.7918803 459781559, -0.9245960638555689, -0.4851452933877 4437, 0.9319676748732453, -72.06095861548073, -2.6060644001377 79, -0.7341809415012422, -0.8051452590888158 3, 0.7415288286242665, -0.6015827589088303]
```

[266]:
```
res = Base.sum([(exp(-x[i+1]))/(x[i] + 10) for i in 1:249])
```

[266]:
```
0.0002052102255397825
```

[270]:
```
b5 = []

for i in y
    if i > 600
        push!(b5, i)
    end
end

print(b5, '\n', findall(y .> 600))
```

```
Any[947, 848, 878, 694, 656, 787, 805, 649, 793, 674, 919, 787, 894, 632, 671, 753, 744, 697, 946, 974, 831, 771, 868, 718, 705, 906, 856, 610, 709, 866, 947, 763, 970, 699, 906, 967, 902, 694, 875, 808, 986, 947, 956, 674, 939, 656, 975, 968, 631, 769, 995, 951, 711, 759, 604, 688, 843, 870, 804, 895, 663, 926, 895, 852, 971, 894, 773, 971, 784, 928, 859, 945, 972, 693, 844, 828, 646, 927, 846, 999, 622, 805, 863, 637, 786, 633, 914, 695, 999, 918, 639, 899, 808, 626, 972, 947, 987, 657, 911, 883, 926, 808, 715, 794, 739]
[1, 3, 8, 9, 11, 13, 14, 21, 22, 23, 24, 28, 33, 36, 38, 40, 43, 47, 48, 50, 56, 57, 58, 59, 63, 64, 66, 67, 68, 69, 70, 71, 74, 77, 81, 86, 92, 94, 95, 97, 98, 99, 102, 107, 108, 109, 110, 111, 116, 118, 121, 122, 125, 127, 128, 129, 133, 134, 137, 139, 140, 144, 147, 148, 149, 152, 153, 154, 156, 157, 159, 161, 164, 168, 170, 175, 176, 178, 181, 185, 186, 188, 196, 197, 198, 202, 204, 205, 206, 207, 211, 217, 221, 223, 224, 226, 227, 230, 232, 233, 237, 240, 244, 246, 247]
```

Рис. 4.15: Задание 3 (14.3-5).

[272]:
```
indexes_y = findall(y .> 600)
b6 = []

for i in indexes_y
    push!(b6, x[i])
end

print(b6)
```

```
Any[81, 343, 286, 73, 848, 549, 700, 974, 506, 423, 811, 264, 87, 557, 785, 462, 387, 95, 188, 917, 862, 794, 949, 835, 745, 322, 883, 691, 786, 772, 159, 954, 436, 303, 884, 356, 459, 3, 957, 592, 35, 483, 507, 689, 864, 461, 106, 52, 772, 642, 915, 611, 55, 652, 935, 964, 857, 149, 34, 239, 170, 266, 949, 278, 202, 639, 574, 975, 429, 137, 83, 592, 841, 90, 339, 112, 896, 787, 650, 376, 821, 330, 730, 174, 648, 75, 764, 554, 800, 864, 147, 990, 505, 877, 759, 462, 803, 556, 617, 693, 31, 847, 208, 818, 933]
```

[274]:
```
x_mean = sum(x) / length(x)
b7 = []

for i in 1:250
    push!(b7, (abs(x[i] - x_mean))^0.5)
end

print(b7)
```

```
Any[20.841017249644988, 22.368459938046694, 13.128137720179506, 22.613005107680845, 14.674740202129643, 11.253977074794495, 17.82279439369708, 15.144239829057117, 21.032070749215535, 22.546574019127604, 18.238749957165538, 21.595092030499721, 5.801034339052037, 13.588671752603345, 20.008698108572668, 9.86650900774395, 17.990330736255941, 19.27827793139211, 15.545160018475205, 15.154273324709438, 21.416162121164476, 3.0574499178236683, 9.609786678173453, 17.194534015203786, 14.224907732565438, 19.00915568877271, 22.30130041051418, 15.853958496224214, 11.025959654747171, 19.035440630571177, 9.036149622488555, 18.128761678614456, 20.69656976409376, 8.523614256874842, 12.741585458646815, 6.453836068571935, 16.752671428726025, 16.42108400806719, 7.16575187960063, 7.303971522397932, 19.069661769417937, 21.740561170310208, 11.329077632358247, 6.507534095185361 5, 14.8543596294152, 20.330961610312 48, 20.502390104570733, 18.092760983332532, 14.640628401813904, 20.041257445579607, 19.348074839631977, 15.916909247715147, 18.832206455962616, 16.320171567725627, 7.11702184960184, 18.618592857678586, 16.692872730599728, 20.824312713748803, 17.878814278357503, 11.195892103803072, 14.468863120508612, 16.9513421297548, 15.154273324709438, 13.904963142705556, 17.272753110028521, 19.174253570869453, 13.253376928164386, 16.45150440904713 83, 16.020362043349 71, 18.877181993083607, 20.944020626422237, 20.655556153248455, 15.989121301685094, 8.907749435182826, 22.479946619153 7, 13.4665511546201, 14.57216524748467, 9.780184047347987, 4.080686216802272, 20.84830928396833, 19.200312497456 91, 16.51217732462924, 19.009155688772 71, 14.651006791343729, 5.626011020252275, 12.623311768311831, 9.129512582822 187, 8.924796916456982, 11.547640451624737, 5.598928468912597, 14.305663214265882, 7.506530490179864, 12.179819374686964, 22.635105477 9959, 21.015518075936175, 18.67222536282165, 8.755112791963336, 21.916842838328698, 5.687530219699932, 19.535915642733514, 6.192576200580818, 2.889290570364 974, 16.381330837267157, 4.8633321909982 71, 20.848309283968 33, 9.661676873089887, 13.177708450257 96, 18.672225362821 65, 7.372109603091 91, 20.232350333068077, 21.525519738208413, 22.479946619 1537, 11.209460290308362, 11.762142661947268, 14.503378916652 49, 16.020362043349 71, 10.646501772882958, 11.253977074794495, 9.074579880082 6, 10.067174380132688, 19.991298106926426, 9.780184047347987, 19.991298106926426, 20.720714273402834, 21.455721847561 18, 13.089996180 28974, 11.689824635126056, 20.485409441844 21, 21.181406940994265, 11.590858466912 62, 3.413502599969721, 16.104285144023002, 18.483830771785378, 19.140219 4344788, 16.960306601002237, 16.135302910078 88, 21.939644482078553, 7.723470722414894, 16.623717995683155, 18.583541104967 05, 13.128137720179506, 21.252105777016446, 12.870586622217344, 15.790756789970516, 7.165751879600 63, 21.455721847563 18, 20.824312713748803, 15.406102686922477, 17.701638342255215, 16.593613229191526, 11.209460290308362, 11.119892088580731, 7.658459375096276, 21.439496262739013, 18.266143544820 84, 9.292362455263998, 19.451169630641 75, 9.728925942774 98, 20.792979584465 52, 20.792979584465 52, 8.755112791963336, 17.738432850734025, 13.392087215964507, 18.045830543369 29, 13.698613068482 52, 22.590883116868184, 21.756562228440412, 20.623966640779848, 11.284857110304939, 13.279608427962021, 11.074836341905918, 20.984470448405 41, 6.296983404774 07, 11.859679590950172, 20.083525586908 29, 19.510304969425775, 6.682215201563 03, 16.481868826076735, 6.807936544945169, 20.387545217607734, 11.603964839657179, 15.088141038577286, 12.261647523885197, 9.831174904354008, 11.804575384146604, 17.482905936943094, 5.945418404115892, 13.614257232768887, 19.757226526008147, 19.561492785572373, 15.242965590724134, 13.466551154620 1, 15.210128204587887, 20.575422231390537, 6.272798418568858, 14.651006791343729, 18.475605538114305, 11.517464998861513, 18.725704259119336, 11.284857110304939, 15.759060885725392, 20.984470448405 41, 19.561492785572373, 15.768703180667712, 6.217073266417249, 16.871632997430925, 18.672225362821 65, 15.187231479107707, 18.202966791157973, 7.787939393703577, 19.192394326920233, 11.985491228981816, 16.051541982002853, 5.687530219699932, 21.362303246607095, 17.540011402504845, 21.786509587357035, 20.509802534398034, 8.021720513705 27, 19.688880110355671, 3.216830738475364 3, 18.832206455962616, 19.017150154531567, 15.609356168657 31, 21.671455880950 87, 7.303971522397932, 16.960306601002237, 16.74395790629914, 15.853958496224214, 6.375892094444513, 14.129826608985 69, 10.082261651038424, 13.328615832111002, 19.714258799153 47, 4.863332190998271, 1.628496238865796, 22.007907669744526, 11.943701268869715, 10.067174380132688, 18.211315163930365, 8.812037221891432, 13.241903186475875, 15.210128204587887, 17.531343359822714, 11.284857110304939, 17.396896274910 65, 20.436535909823 96, 16.451504490471383, 1.829754081837216 9, 17.194534015203786]
```

[278]:
```
y_max = maximum(y)
count = 0

for i in y
    if abs(i - y_max) <= 200
        count += 1
    end
end

print(count)
```

```
61
```

Рис. 4.16: Задание 3 (14.6-8).

[282]:
```
count_even = 0

for i in x
    if i % 2 == 0
        count_even += 1
    end
end

print("Чётных элементов ", count_even, ", нечётных элементов ", 250 - count_even)
```

```
Чётных элементов 125, нечётных элементов 125
```

[284]:
```
count_seven = 0

for i in x
    if i % 7 == 0
        count_seven += 1
    end
end

print(count_seven, " элементов кратно семи")
```

```
39 элементов кратно семи
```

[286]:
```
indexes_y = sortperm(y)
x_sorted_by_y = x[indexes_y]

print(x_sorted_by_y)
```

```
Any[527, 92, 597, 803, 931, 788, 612, 18, 830, 656, 518, 786, 305, 796, 284, 154, 870, 950, 433, 833, 153, 154, 365, 10, 720, 898, 743, 870, 588, 262, 539, 334, 539, 367, 904, 483, 377, 235, 418, 414, 388, 593, 745, 575, 255, 141, 757, 228, 86, 102, 641, 75, 217, 334, 5, 898, 844, 83, 681, 839, 381, 773, 414, 849, 532, 915, 897, 736, 402, 950, 343, 353, 249, 4, 715, 388, 7, 49, 115, 611, 464, 512, 469, 887, 267, 547, 595, 340, 480, 703, 936, 903, 42, 477, 284, 555, 388, 476, 451, 344, 283, 55, 730, 306, 811, 390, 659, 746, 59, 642, 823, 638, 637, 967, 313, 184, 240, 300, 879, 658, 256, 942, 432, 247, 484, 264, 10, 576, 125, 866, 771, 560, 864, 985, 988, 610, 422, 464, 336, 473, 382, 15, 566, 641, 301, 935, 691, 821, 877, 772, 557, 75, 174, 147, 896, 974, 848, 461, 556, 170, 785, 423, 689, 964, 90, 73, 3, 554, 95, 303, 745, 786, 55, 208, 835, 933, 387, 462, 652, 954, 642, 794, 574, 429, 648, 549, 264, 506, 818, 34, 700, 330, 592, 505, 847, 112, 862, 857, 339, 650, 343, 278, 883, 83, 730, 772, 949, 149, 957, 286, 693, 87, 639, 239, 949, 990, 459, 322, 884, 617, 764, 864, 811, 266, 31, 787, 137, 864, 592, 188, 81, 159, 483, 462, 611, 507, 356, 52, 436, 202, 975, 841, 759, 917, 106, 35, 803, 915, 376, 800]
```

[288]:
```
b10 = reverse(last(sort(x), 10))

print(b10)
```

```
Any[990, 988, 985, 975, 974, 967, 964, 957, 954, 950]
```

[290]:
```
b11 = unique(x)

print(b11)
```

```
Any[81, 15, 343, 4, 300, 642, 833, 286, 73, 7, 848, 49, 549, 700, 115, 418, 839, 887, 757, 745, 974, 506, 423, 811, 313, 154, 18, 264, 637, 153, 597, 844, 87, 588, 353, 557, 796, 785, 464, 462, 879, 988, 387, 473, 736, 102, 95, 188, 301, 917, 141, 262, 870, 249, 566, 862, 794, 949, 835, 390, 306, 228, 322, 217, 883, 691, 786, 772, 159, 954, 942, 771, 436, 10, 334, 303, 611, 532, 950, 884, 788, 730, 547, 356, 432, 595, 382, 484, 720, 459, 367, 3, 957, 864, 592, 35, 483, 897, 477, 507, 247, 539, 422, 689, 461, 106, 52, 641, 377, 305, 402, 433, 414, 915, 86, 55, 344, 652, 935, 964, 381, 527, 256, 857, 149, 803, 255, 34, 575, 239, 170, 967, 681, 266, 278, 202, 240, 639, 574, 975, 849, 429, 137, 610, 83, 830, 336, 841, 703, 5, 42, 90, 388, 339, 638, 75, 555, 656, 112, 896, 560, 787, 469, 931, 650, 743, 365, 612, 376, 821, 480, 330, 125, 898, 283, 284, 92, 476, 174, 648, 866, 267, 764, 554, 800, 746, 184, 576, 147, 659, 773, 59, 823, 990, 936, 451, 903, 505, 877, 759, 985, 235, 556, 715, 617, 693, 904, 518, 31, 658, 847, 593, 340, 208, 818, 933, 512]
```

Рис. 4.17: Задание 3 (14.9-13).

4. Создадим массив squares, в котором будут храниться квадраты всех целых чисел от 1 до 100 (рис. 4.18).

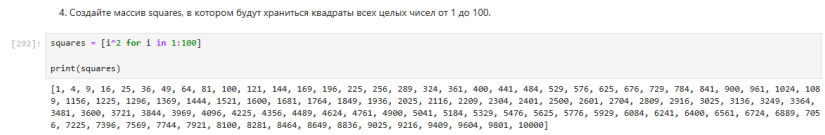

Рис. 4.18: Задание 4

5. 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. Получии срез массива с 89-го до 99-го элемента включительно, содержащий наименьшие простые числа (рис. 4.19).

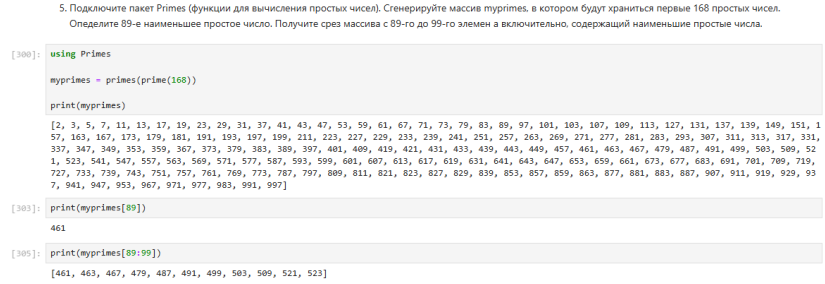

Рис. 4.19: Задание 5

6. Вычислим выражения (рис. 4.20).

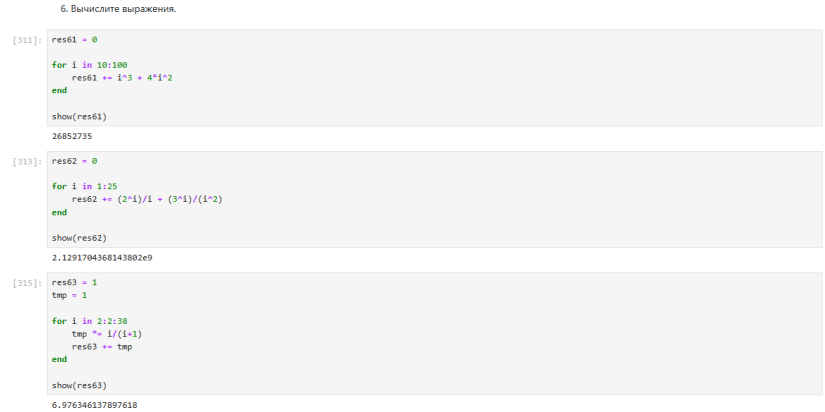

Рис. 4.20: Задание 6

5 Выводы

Изучила несколько структур данных, реализованных в Julia, научилась применять их и операции над ними для решения задач.