Презентация по лабораторной работе №2

Дисциплина: Компьютерный практикум по статистическому анализу данных

Лобанова П.И.

24 сентября 2025

Российский университет дружбы народов, Москва, Россия

Информация

Докладчик

- Лобанова Полина Иннокентьевна
- Учащаяся на направлении “Фундаментальная информатика и информационные технологии”
- Студентка группы НФИбд-02-22
- polla-2004@mail.ru

Цель

Цель

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

Задание

Задание

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

Выполнение

Выполнение

```
*[2]: #пустой кортеж
      ()

[2]:  ()

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

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

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

[4]:  (1, 2, 3)

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

[6]:  (1, 2.0, "tmp")

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

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

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

[9]:  3

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

Рис. 1: *Примеры создания кортежей и операций над ними*

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

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

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

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

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

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

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

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

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

[33]: true
```

Рис. 2: *Примеры создания словарей и операций над ними*

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

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

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

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

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

```
[65]: false
```

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

```
[66]: true
```

Рис. 3: *Примеры создания множеств и операций над ними*

```
[73]: # создание пустого массива с абстрактным типом:
      enpty_array_1 = []

[73]: Any[]

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

[76]: Float64[]

[77]: # вектор-столбец:
      a = [1, 2, 3]

[77]: 3-element Vector{Int64}:
       1
       2
       3

[78]: # вектор-строка:
      b = [1 2 3]

[78]: 1×3 Matrix{Int64}:
       1  2  3

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

[79]: 3×3 Matrix{Int64}:
       1  4  7
       2  5  8
       3  6  9
```

Рис. 4: *Примеры создания массивов и операций над ними*

[115]:

```
#1
A = Set([0, 3, 4, 9]);
B = Set([1, 3, 4, 7]);
C = Set([0, 1, 2, 4, 7, 8, 9]);
x1= intersect(A,B);
x2 = union(x1,A);
x3 = intersect(x2, B);
x4 = union(x3, A);
x5 = intersect(x4, C);
x6 = union(x5, B);
P = intersect(x6, C)
```

[115]:

```
Set{Int64} with 5 elements:
  0
  4
  7
  9
  1
```

Рис. 5: *Задание 1*

```
#2
D = Set(["ad", "fg"]);
E = Set(["er", "ad"]);
x7 = union(D, E);
x8 = intersect(D, E);
x9 = setdiff(D, E);
print("Объединение: ", x7, '\n', "Пересечение: ", x8, '\n', "Разность: ", x9)
```

```
Объединение: Set(["ad", "er", "fg"])
Пересечение: Set(["ad"])
Разность: Set(["fg"])
```

Рис. 6: *Задание 2*

массив (1, 2, 3, … N − 1, N), N = 25;

```
#3
#3.1
N = 25;
ar1 = [i for i in 1:N];
ar2 = collect(1:N);
print(ar1, '\n', ar2)
```

```
[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]
[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]
```

Рис. 7: *Задание 3.1*

массив (N, N − 1 ... , 2, 1);

```
#3.2
ar3 = [i for i in N:-1:1];
ar4 = collect(N:-1:1);
print(ar3, '\n', ar4)
```

```
[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]
[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]
```

Рис. 8: *Задание 3.2*

массив (1, 2, 3, ... , N − 1, N, N − 1, ... , 2, 1);

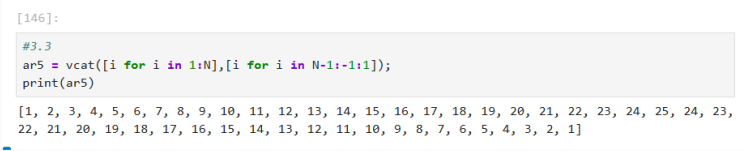


Рис. 9: *Задание 3.3*

массив с именем tmp вида (4, 6, 3);

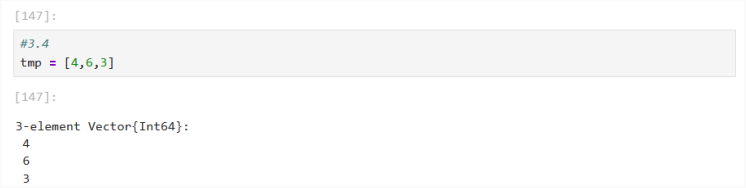


Рис. 10: *Задание 3.4*

массив, в котором первый элемент массива tmp повторяется 10 раз;

```
[153]:
#3.5
ar6 = fill(tmp[1], (1, 10))
[153]:
1×10 Matrix{Int64}:
 4  4  4  4  4  4  4  4  4  4
```

Рис. 11: *Задание 3.5*

массив, в котором все элементы массива tmp повторяются 10 раз;

[154]:

```
#3.6
ar7=vcat(fill(tmp[1], (1, 10)), fill(tmp[2], (1, 10)), fill(tmp[3], (1, 10)))
```

[154]:

```
3×10 Matrix{Int64}:
 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
```

Рис. 12: *Задание 3.6*

массив, в котором первый элемент массива tmp встречается 11 раз, второй элемент — 10 раз, третий элемент — 10 раз;

[157]:

```
#3.7
ar8=vcat(fill.(tmp, [11,10,10]))
```

[157]:

```
3-element Vector{Vector{Int64}}:
 [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]
```

Рис. 13: *Задание 3.7*

массив, в котором первый элемент массива tmp встречается 10 раз подряд, второй элемент — 20 раз подряд, третий элемент — 30 раз подряд;

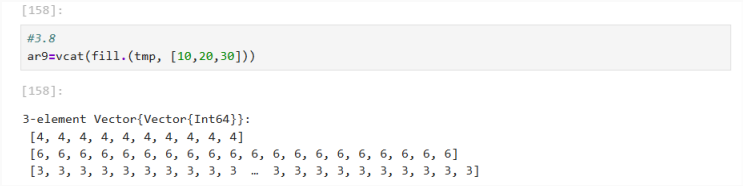


Рис. 14: *Задание 3.8*

массив из элементов вида 2^tmp[i], i = 1, 2, 3, где элемент 2^tmp[3] встречается 4 раза;
посчитала в полученном векторе, сколько раз встречается цифра 6, и вывела это значение на экран;

[167]:

```
#3.9
tmp1 = [2^(tmp[i]) for i in 1:3];
ar10 = vcat(fill.(tmp1, [3,2,4]));
count=0
for i in ar10
    if i==6
        count+=1
    else
        continue
    end
end
print(count)
```

```
0
```

Рис. 15: *Задание 3.9*

вектор значений y=e^x cos(x) в точках x = 3, 3.1, 3.2, ... , 6, нашла среднее значение y;

[186]:

```
#3.10
y = [exp(x)*cos(x) for x in 3:0.1:6]
sum=0
for i in y
    sum+=i
end
avg=sum/length(y)
```

[186]:

```
53.11374594642971
```

Рис. 16: *Задание 3.10*

вектор вида (x^i, y^j), x = 0.1, i = 3, 6, 9, ... , 36, y = 0.2, j = 1, 4, 7, ... , 34;

[199]:

```
#3.11
ar11=[]
for j in 1:3:34
    i=j+2
    push!(ar11, (0.1^i, 0.2^j))
end
print(ar11)
```

```
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, 6.5536000000000055e-12), (1.0000000000000012e-21, 5.2428800000000056e-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)]
```

Рис. 17: *Задание 3.11*

вектор с элементами 2^i/i, i=1,2,...,M, M = 25;

[200]:

```
#3.12
ar12=[(2^i)/i for i in 1:25]
```

[200]:

```
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
```

Рис. 18: *Задание 3.12*

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

[209]:

```
#3.13
ar13=[]
for i in 1:30
    push!(ar13, "fn$i")
end
ar13
```

[209]:

```
30-element Vector{Any}:
 "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"
```

Рис. 19: *Задание 3.13*

векторы x=(x1, x2, ..., xn) и y=(y1, y2, ..., yn) целочисленного типа длины n = 250 как случайные выборки из совокупности 0, 1, ... , 999; на его основе:

– сформировала вектор (y2-x1, ..., yn - x n-1);

Рис. 20: *Задание 3.14*

– сформировала вектор (x1+2x2-x3, x2+2x3-x4, ..., x n-2 +2x n-1-xn);

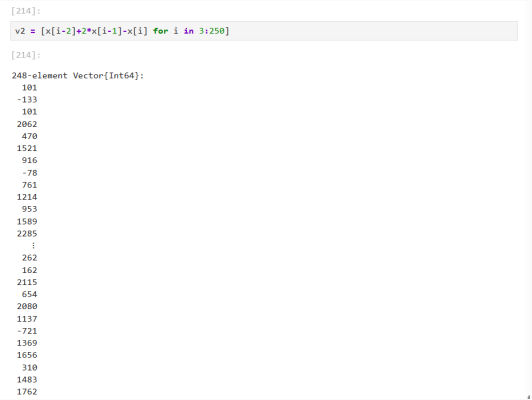


Рис. 21: *Задание 3.15*

- сформируйте вектор $\left(\frac{\sin(y_1)}{\cos(x_2)}, \frac{\sin(y_2)}{\cos(x_3)}, \dots, \frac{\sin(y_{n-1})}{\cos(x_n)}\right)$;
- вычислите $\sum_{i=1}^{n-1} \frac{e^{-x_{i+1}}}{x_i + 10}$;

Рис. 22: *Задачи*

[215]:

```
v3 = [(sin(y[i-1]))/(cos(x[i])) for i in 2:250]
```

[215]:

```
249-element Vector{Float64}:
 -1.754722687152709
  0.9924804445211151
  0.8887114139147112
  2.2097483587123703
  1.2926131931680243
  0.8865820542365037
 -1.1181275586399602
  1.2556847312195138
 -0.3504773299757346
  0.3233601793275197
 -0.6605551156470216
 -4.110448071676383
 -1.3054656540484193
  ⋮
  0.1366529552109628
 -0.9904019338058403
 -0.49407478784638204
 -0.26896935770095953
 -1.1523421912916259
 -2.2997286020434804
  0.5028973046701645
 -0.03779959921553782
  0.6240594659965716
  0.6037466961066857
  8.521144605305865
 -0.7578279342271558
```

Рис. 23: *Задание 3.16*

[220]:

```
sum=0
for i in 1:249
    sum+=(exp(-x[i+1]))/x[i]+10
    end
print(sum)
```

```
2490.000551954167
```

Рис. 24: *Задание 3.17*

– выбрала элементы вектора y, значения которых больше 600, и вывела на экран;
определила индексы этих элементов;

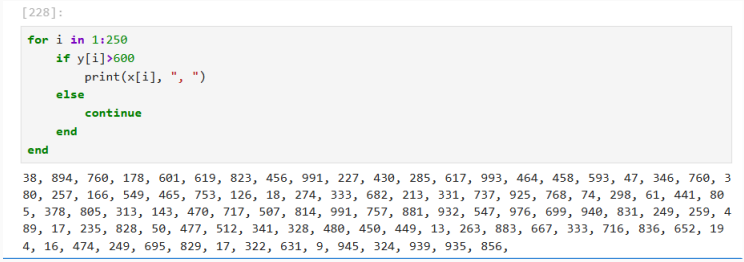
```
for i in 1:250
    if y[i]>600
        print(y[i], ", ")
    else
        continue
    end
end

print('\n',"Индексы: ", findall(y .> 600))
```

```
765, 823, 969, 861, 961, 725, 904, 605, 787, 697, 607, 993, 883, 893, 715, 980, 874, 838, 916, 707,
931, 755, 917, 788, 768, 715, 861, 767, 910, 790, 842, 991, 754, 919, 851, 708, 804, 914, 834, 700,
694, 849, 811, 667, 944, 612, 615, 839, 847, 871, 862, 977, 639, 720, 901, 967, 832, 872, 707, 637,
912, 868, 824, 888, 943, 949, 728, 737, 624, 983, 972, 663, 867, 934, 873, 842, 616, 614, 724, 764,
850, 630, 858, 762, 603, 766, 793, 801, 979, 952, 964, 993, 831, 622, 706,
Индексы: CartesianIndex{2}[CartesianIndex(1, 1), CartesianIndex(1, 5), CartesianIndex(1, 7), Cartes
ianIndex(1, 9), CartesianIndex(1, 10), CartesianIndex(1, 11), CartesianIndex(1, 20), CartesianIndex
(1, 22), CartesianIndex(1, 24), CartesianIndex(1, 31), CartesianIndex(1, 32), CartesianIndex(1, 3
3), CartesianIndex(1, 36), CartesianIndex(1, 37), CartesianIndex(1, 40), CartesianIndex(1, 41), Car
tesianIndex(1, 45), CartesianIndex(1, 46), CartesianIndex(1, 47), CartesianIndex(1, 49), CartesianI
ndex(1, 52), CartesianIndex(1, 55), CartesianIndex(1, 58), CartesianIndex(1, 59), CartesianIndex(1,
62), CartesianIndex(1, 63), CartesianIndex(1, 64), CartesianIndex(1, 67), CartesianIndex(1, 68), Ca
rtesianIndex(1, 71), CartesianIndex(1, 72), CartesianIndex(1, 73), CartesianIndex(1, 75), Cartesian
Index(1, 76), CartesianIndex(1, 78), CartesianIndex(1, 79), CartesianIndex(1, 85), CartesianIndex
(1, 87), CartesianIndex(1, 91), CartesianIndex(1, 92), CartesianIndex(1, 94), CartesianIndex(1, 9
6), CartesianIndex(1, 98), CartesianIndex(1, 100), CartesianIndex(1, 101), CartesianIndex(1, 102),
CartesianIndex(1, 103), CartesianIndex(1, 106), CartesianIndex(1, 110), CartesianIndex(1, 111), Car
tesianIndex(1, 113), CartesianIndex(1, 114), CartesianIndex(1, 115), CartesianIndex(1, 117), Cartes
ianIndex(1, 119), CartesianIndex(1, 121), CartesianIndex(1, 125), CartesianIndex(1, 128), Cartesian
Index(1, 135), CartesianIndex(1, 141), CartesianIndex(1, 142), CartesianIndex(1, 143), CartesianInd
ex(1, 145), CartesianIndex(1, 146), CartesianIndex(1, 150), CartesianIndex(1, 159), CartesianIndex
(1, 165), CartesianIndex(1, 166), CartesianIndex(1, 167), CartesianIndex(1, 168), CartesianIndex(1,
169), CartesianIndex(1, 172), CartesianIndex(1, 174), CartesianIndex(1, 181), CartesianIndex(1, 18
2), CartesianIndex(1, 184), CartesianIndex(1, 186), CartesianIndex(1, 187), CartesianIndex(1, 192),
CartesianIndex(1, 193), CartesianIndex(1, 194), CartesianIndex(1, 201), CartesianIndex(1, 211), Car
tesianIndex(1, 215), CartesianIndex(1, 222), CartesianIndex(1, 225), CartesianIndex(1, 228), Cartes
ianIndex(1, 229), CartesianIndex(1, 233), CartesianIndex(1, 238), CartesianIndex(1, 240), Cartesian
Index(1, 241), CartesianIndex(1, 242), CartesianIndex(1, 245), CartesianIndex(1, 249)]
```

Рис. 25: *Задание 3.18*

– определила значения вектора x, соответствующие значениям вектора y, значения которых больше 600 (под соответствием понимается расположение на аналогичных индексных позициях);



[228]:

```
for i in 1:250
    if y[i]>600
        print(x[i], ", ")
    else
        continue
    end
end
```

```
38, 894, 760, 178, 601, 619, 823, 456, 991, 227, 430, 285, 617, 993, 464, 458, 593, 47, 346, 760, 3
80, 257, 166, 549, 465, 753, 126, 18, 274, 333, 682, 213, 331, 737, 925, 768, 74, 298, 61, 441, 80
5, 378, 805, 313, 143, 470, 717, 507, 814, 991, 757, 881, 932, 547, 976, 699, 940, 831, 249, 259, 4
89, 17, 235, 828, 50, 477, 512, 341, 328, 480, 450, 449, 13, 263, 883, 667, 333, 716, 836, 652, 19
4, 16, 474, 249, 695, 829, 17, 322, 631, 9, 945, 324, 939, 935, 856,
```

Рис. 26: *Задание 3.19*

– сформируйте вектор $(|x_1 - \overline{x}|^{\frac{1}{2}}, |x_2 - \overline{x}|^{\frac{1}{2}}, \dots, |x_n - \overline{x}|^{\frac{1}{2}})$, где $\overline{x}$ обозначает среднее значение вектора $x = (x_1, x_2, \dots, x_n)$;

Рис. 27: *Задачи*

[230]:

```
sum=0
for i in x
    sum+=i
end
avg=sum/length(x)
v4=[(abs(x[i]-avg))^0.5 for i in 1:250]
```

[230]:

```
250-element Vector{Float64}:
 21.590460856591275
 20.424201330774235
 19.827960056445544
  7.883400281604381
 19.744670166908335
 18.3343393663366
 15.995374331349673
 18.36159034506543
 18.059568101147935
  9.841341371987864
 10.716902537580529
 10.993270668913778
 20.294137084389668
  ⋮
  6.697163578710018
 20.99647589477815
 13.421922366039823
 20.853105284345542
 19.54860608841459
 21.404392072656492
 20.756974731400526
  6.546143903092871
 11.45198672720153
 21.88268722072314
 18.75771841136336
 20.70874211534829
```

Рис. 28: *Задание 3.20*

– определила, сколько элементов вектора y отстоят от максимального значения не более, чем на 200;

[234]:

```
max1 = maximum(y)
count=0
for i in 1:250
    if max1 - y[i] < 200
        count+=1
    else
        continue
    end
end
print(count)
```

```
55
```

Рис. 29: *Задание 3.21*

– определила, сколько чётных и нечётных элементов вектора x;

[239]:

```
count_ch=0
for i in 1:250
    if x[i]%2 ==0
        count_ch+=1
    else
        continue
    end
end
print(count_ch)
```

```
123
```

[240]:

```
count_nch=0
for i in 1:250
    if x[i]%2 !=0
        count_nch+=1
    else
        continue
    end
end
print(count_nch)
```

```
127
```

Рис. 30: *Задание 3.22*

– определила, сколько элементов вектора x кратны 7;

[242]:

```
count_7=0
for i in 1:250
    if x[i]%7 ==0
        count_7+=1
    else
        continue
    end
end
print(count_7)
```

```
30
```

Рис. 31: *Задание 3.23*

– вывела элементы вектора x, которые входят в десятку наибольших (top-10);

```
[253]:
reverse(last(sort(x,dims=2), 10))
```

```
[253]:
10-element Vector{Int64}:
 993
 993
 991
 991
 984
 983
 976
 973
 954
 951
```

Рис. 32: *Задание 3.24*

– сформировала вектор, содержащий только уникальные (неповторяющиеся) элементы вектора x.

```
[254]:
unique(x)

[254]:
218-element Vector{Int64}:
  38
  87
 111
 442
 894
 168
 760
 167
 178
 601
 619
 625
 916
   ⋮
 724
 104
 945
 793
 324
 939
  46
 935
 373
 983
 856
 933
```

Рис. 33: *Задание 3.25*

[255]:

```
#4
squares = [i^2 for i in 1:100]
```

[255]:

```
100-element Vector{Int64}:
     1
     4
     9
    16
    25
    36
    49
    64
    81
   100
   121
   144
   169
     ⋮
  7921
  8100
  8281
  8464
  8649
  8836
  9025
  9216
  9409
  9604
  9801
 10000
```

Рис. 34: *Задание 4*

```
import Primes
```

[287]:

```
myprimes = Primes.primes(Primes.prime(168))
myprimes[89]
```

[287]:

```
461
```

[288]:

```
Primes.prime(89)
```

[288]:

```
461
```

[289]:

```
myprimes[89:99]
```

[289]:

```
11-element Vector{Int64}:
 461
 463
 467
 479
 487
 491
 499
 503
 509
 521
 523
```

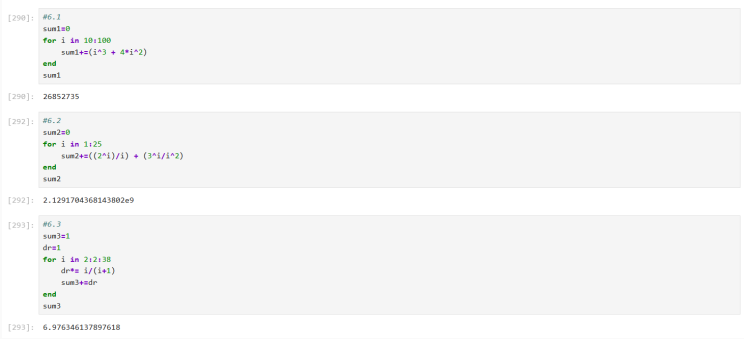
Рис. 35: *Задание 5*

6.1) $\sum_{i=10}^{100} (i^3 + 4i^2)$;

6.2) $\sum_{i=1}^{M} \left(\frac{2^i}{i} + \frac{3^i}{i^2}\right), M = 25$;

6.3) $1 + \frac{2}{3} + \left(\frac{2}{3}\frac{4}{5}\right) + \left(\frac{2}{3}\frac{4}{5}\frac{6}{7}\right) + \cdots + \left(\frac{2}{3}\frac{4}{5}\cdots\frac{38}{39}\right)$.

Рис. 36: *Задачи*



```
[290]: #6.1
       sum1=0
       for i in 10:100
           sum1+=(i^3 + 4*i^2)
       end
       sum1
```

```
[290]: 26852735
```

```
[292]: #6.2
       sum2=0
       for i in 1:25
           sum2+=((2^i)/i) + (3^i/i^2)
       end
       sum2
```

```
[292]: 2.1291704368143802e9
```

```
[293]: #6.3
       sum3=1
       dr=1
       for i in 2:2:38
           dr*= i/(i+1)
           sum3+=dr
       end
       sum3
```

```
[293]: 6.976346137897618
```

Рис. 37: *Задание 6*

Вывод

Вывод

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