Отчёт по лабораторной работе №2

Структуры данных

Гань Чжаолун

Цель работы

Изучение структур данных, реализованных в Julia. Научиться применять их и операции над ними для решения задач.

Выполнение лабораторной работы

2.2.1 Кортежи

Кортеж (Tuple) — структура данных (контейнер) в виде неизменяемой индексируемой последовательности элементов какого-либо типа (элементы индексируются с единицы). Синтаксис определения кортежа: (element1, element2, ...)

```
# пустой кортеж:
()
```

```
()
```

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

```
("Python", "Julia", "R")
```

```
# кортеж из целых чисел
x1 = (1, 2, 3)
```

```
(1, 2, 3)
```

```
# кортеж из элементов разных типов
x2 = (1, 2.0, "tmp")
```

```
(1, 2.0, "tmp")
```

```
# именованный кортеж:
x3 = (a=2, b=1+2)
```

```
(a = 2, b = 3)
```

Рисунок 1. Примеры создания кортежей

```
# длина кортежа x1
length(x2)
```

```
3
```

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

```
(1, 2.0, "tmp")
```

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

```
5
```

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

```
(2, 3, 3)
```

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

```
(true, false)
```

Рисунок 2. Примеры операций над кортежами

2.2.2 Словари

Словарь — неупорядоченный набор связанных между собой по ключу данных. Синтаксис определения словаря: Dict(key1 => value1, key2 => value2, ...)

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

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

```
# вывести ключи словаря:
keys(phonebook)
```

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

```
# вывести значения элементов словаря:
values(phonebook)
```

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

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

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

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

```
true
```

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

```
"555-3344"
```

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

```
("867-5309", "333-5544")
```

Рисунок 3. Примеры инициализации и операций над словарями

```
# 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)
```

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

Рисунок 4. Объединение словарей

2.2.3 Множества

Множество, как структура данных в Julia, соответствует множеству, как математическому объекту, то есть является неупорядоченной совокупностью элементов какого-либо типа. Возможные операции над множествами: объединение, пересечение, разность; принадлежность элемента множеству. Синтаксис

определения множества: Set([itr]), где itr — набор значений, сгенерированных данным итерируемым объектом или пустое множество.

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

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

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

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

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

```
false
```

```
# проверка эквивалентности 3 и 4 множества
S3 = Set([1,2,2,3,1,2,3,2,1]);
S4 = Set([2,3,1]);
issetequal(S3,S4)
```

```
true
```

```
# объединение множеств:
C=union(S1,S2)
```

```
Set{Int64} with 4 elements:
  4
  2
  3
  1
```

Рисунок 5. Примеры множеств и операций над ними (Часть 1)

```
# пересечение множеств:
D = intersect(S1,S3)
```

```
Set{Int64} with 2 elements:
  2
  1
```

```
# разность множеств:
E = setdiff(S3,S1)
```

```
Set{Int64} with 1 element:
  3
```

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

```
true
```

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

```
Set{Int64} with 4 elements:
  2
  99
  3
  1
```

```
# удаление последнего элемента множества:
pop!(S4)
```

```
2
```

```
S4
```

```
Set{Int64} with 3 elements:
  99
  3
  1
```

Рисунок 6. Примеры множеств и операций над ними (Часть 2)

2.2.4 Массивы

Массив — коллекция упорядоченных элементов, размещённая в многомерной сетке. Векторы и матрицы являются частными случаями массивов. Общий синтаксис одномерных массивов: oarray*name*1 = [element1, element2, ...] oarray*name*2 = [element1 element2 ...]

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

```
Any[]
```

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

```
Int64[]
```

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

```
Float64[]
```

Рисунок 7. Пример объявления массивов (1)

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

```
3-element Vector{Int64}:
 1
 2
 3
```

```
# вектор-строка:
b = [1 2 3]
```

```
1×3 Matrix{Int64}:
 1  2  3
```

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

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

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

```
1×8 Matrix{Float64}:
 0.139052  0.265166  0.145633  0.0896685  …  0.41114  0.893847  0.267533
```

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

```
2×3 Matrix{Float64}:
 0.0355167  0.529554  0.0514481
 0.119321   0.47618   0.465355
```

Рисунок 8. Пример объявления массивов (2)

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

```
4×3×2 Array{Float64, 3}:
[:, :, 1] =
 0.283411  0.218688  0.454782
 0.938875  0.832847  0.877779
 0.703477  0.217353  0.445808
 0.144286  0.915661  0.543622

[:, :, 2] =
 0.366512   0.913371   0.291656
 0.0258695  0.0519026  0.61626
 0.473893   0.0946979  0.357052
 0.595379   0.871507   0.38627
```

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

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

Рисунок 9. Пример объявления массивов (3)

```
# 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]
```

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

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

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

Рисунок 10. Пример объявления массивов (4)

Примеры операций над массивами.

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

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

```
# двумерный массив 2x3 из единиц:
ones(2,3)
```

```
2×3 Matrix{Float64}:
 1.0  1.0  1.0
 1.0  1.0  1.0
```

```
# одномерный массив из 4 нулей:
zeros(4)
```

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

```
# заполнить массив 3x2 цифрами 3.5
fill(3.5,(3,2))
```

```
3×2 Matrix{Float64}:
 3.5  3.5
 3.5  3.5
 3.5  3.5
```

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

```
6×3 Matrix{Int64}:
 1  1  1
 2  2  2
 1  1  1
 2  2  2
 1  1  1
 2  2  2
```

```
repeat([1 2],3,3)
```

```
3×6 Matrix{Int64}:
 1  2  1  2  1  2
 1  2  1  2  1  2
 1  2  1  2  1  2
```

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

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

```
# транспонирование
b'
```

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

```
# транспонирование
c = transpose(b)
```

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

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

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

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

```
10-element Vector{Int64}:
 18
 10
 13
 10
 17
 13
 20
 17
 12
 13
```

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

```
10×2 Matrix{Int64}:
 18  15
 10  10
 13  12
 10  20
 17  15
 13  13
 20  16
 17  20
 12  20
 13  11
```

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

```
10×3 Matrix{Int64}:
 18  17  19
 10  14  20
 13  14  14
 10  18  14
 17  18  20
 13  12  12
 20  14  16
 17  14  10
 12  12  17
 13  17  19
```

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

```
3×2 Matrix{Int64}:
 11  10
 10  20
 19  13
```

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

```
3-element Vector{Int64}:
 17
 19
 15
```

```
# сортировка по строкам:
sort(ar,dims=1)
```

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

```
# сортировка по столбцам:
sort(ar,dims=2)
```

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

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

```
10×5 BitMatrix:
 0  1  1  1  1
 0  0  0  1  0
 0  0  0  0  0
 0  0  1  0  1
 0  1  1  1  1
 1  0  0  0  0
 1  1  0  1  1
 1  1  0  0  1
 0  0  0  1  1
 0  0  1  1  0
```

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

```
23-element Vector{CartesianIndex{2}}:
 CartesianIndex(6, 1)
 CartesianIndex(7, 1)
 CartesianIndex(8, 1)
 CartesianIndex(1, 2)
 CartesianIndex(5, 2)
 CartesianIndex(7, 2)
 CartesianIndex(8, 2)
 CartesianIndex(1, 3)
 CartesianIndex(4, 3)
 CartesianIndex(5, 3)
 CartesianIndex(10, 3)
 CartesianIndex(1, 4)
 CartesianIndex(2, 4)
 CartesianIndex(5, 4)
 CartesianIndex(7, 4)
 CartesianIndex(9, 4)
 CartesianIndex(10, 4)
 CartesianIndex(1, 5)
 CartesianIndex(4, 5)
 CartesianIndex(5, 5)
 CartesianIndex(7, 5)
 CartesianIndex(8, 5)
 CartesianIndex(9, 5)
```

Задания для самостоятельного выполнения

```
1. Даны множества: A = {0, 3, 4, 9}, B = {1, 3, 4, 7}, C = {0, 1, 2, 4, 7, 8, 9}.
Найти P = A ∩ B ∪ A ∩ B ∪ A ∩ C ∪ B ∩ C.
```

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

P = union(union(union(intersect(A, B), intersect(A, B)), intersect(A, C)), intersect(B, C))
print(P)
```

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

Рисунок 11. Код и результат Задания 1

2. Приведите свои примеры с выполнением операций над множествами элементов разных типов.

```
# создание двух множеств с элементами разных типов
A = Set([3, 5, 7, 9])
B = Set(["P", "S", "GZL"])

# объединение множеств A и B
C = union(A, B)
```

```
Set{Any} with 7 elements:
  5
  "S"
  7
  "GZL"
  "P"
  9
  3
```

```
# разность множеств B и A
D = setdiff(B, A)
```

```
Set{String} with 3 elements:
  "S"
  "GZL"
  "P"
```

```
# проверка вхождения элементов множества A в множество B
issubset(A, B)
```

```
false
```

```
# добавление элемента в множество A
push!(A, 12)
```

```
Set{Int64} with 5 elements:
  5
  7
  9
  12
  3
```

```
# удаление последнего элемента множества A
pop!(A)
```

```
5
```

Рисунок 12. Примеры операций над множествами разных типов

3.1 массив (1, 2, 3, ... $N - 1$, N), N выберите больше 20;

```
N =24
A = collect(1:N)
print(A)
```

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

Рисунок 13. Код и результат Задания 3.1

3.2 массив (N, $N - 1$ ... , 2, 1), N выберите больше 20;

```
B = [i for i in N:-1:0]
print(B)
```

```
[24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

Рисунок 14. Код и результат Задания 3.2

3.3 массив (1, 2, 3, … , N − 1, N, N − 1, … , 2, 1), N выберите больше 20;

```
C = []
C = [i for i in 1:N]
for i in N-1:-1:1
    push!(C, i)
end
print(C)
```

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

Рисунок 15. Код и результат Задания 3.3

3.4 массив с именем tmp вида (4, 6, 3);

```
tmp = [4, 6, 3]
print(tmp)
```

```
[4, 6, 3]
```

Рисунок 16. Код и результат Задания 3.4

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

```
tmp_3_5 = fill(tmp[1],(1,10))
print(tmp_3_5)
```

```
[4 4 4 4 4 4 4 4 4 4]
```

Рисунок 17. Код и результат Задания 3.5

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

```
tmp_3_6 = repeat(tmp,10)
print(tmp_3_6)
```

```
[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, 4, 6, 3]
```

Рисунок 18. Код и результат Задания 3.6

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

```
tmp_3_7 = vcat(fill(tmp[1],11),fill(tmp[2],10),fill(tmp[3],10))
print(tmp_3_7)
```

```
[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]
```

Рисунок 19. Код и результат Задания 3.7

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

```
tmp_3_8 = vcat(fill(tmp[1],10),fill(tmp[2],20),fill(tmp[3],30))
print(tmp_3_8)
```

```
[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, 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, 3, 3, 3, 3]
```

Рисунок 20. Код и результат Задания 3.8

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

```
tmp_square = [[2^tmp[i] for i in 1:length(tmp)]; fill(2^tmp[3], 3);]

println(tmp_square)

# разделим все числа в массиве, чтобы найти количество цифр 6
count = 0
numbers = []  # Empty array of Ints
for i in 1:1:length(tmp_square)
    while tmp_square[i] != 0
        rem = tmp_square[i] % 10  # Modulo division - get last digit
        push!(numbers, rem)
        tmp_square[i] = div(tmp_square[i],10)
    end
end

# подсчитаем количество цифр 6
for i in 1:1:length(numbers)
    if numbers[i] == 6
        count += 1
    end
end
print(count)
```

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

Рисунок 21. Код и результат Задания 3.9

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

```
using Statistics

y = [exp(i)*cos(i) for i in 3:0.1:6]

mean(y)
```

```
53.11374594642971
```

Рисунок 22. Код и результат Задания 3.10

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

```
x = 0.1
y = 0.2
vector_1 = [[x^i y^j] for i = 3:3:36, j in 1:3:34]
```

```
12×12 Matrix{Matrix{Float64}}:
 [0.001 0.2]    [0.001 0.0016]    [0.001 1.28e-5]    …  [0.001 1.71799e-24]
 [1.0e-6 0.2]   [1.0e-6 0.0016]   [1.0e-6 1.28e-5]       [1.0e-6 1.71799e-24]
 [1.0e-9 0.2]   [1.0e-9 0.0016]   [1.0e-9 1.28e-5]       [1.0e-9 1.71799e-24]
 [1.0e-12 0.2]  [1.0e-12 0.0016]  [1.0e-12 1.28e-5]      [1.0e-12 1.71799e-24]
 [1.0e-15 0.2]  [1.0e-15 0.0016]  [1.0e-15 1.28e-5]      [1.0e-15 1.71799e-24]
 [1.0e-18 0.2]  [1.0e-18 0.0016]  [1.0e-18 1.28e-5]   …  [1.0e-18 1.71799e-24]
 [1.0e-21 0.2]  [1.0e-21 0.0016]  [1.0e-21 1.28e-5]      [1.0e-21 1.71799e-24]
 [1.0e-24 0.2]  [1.0e-24 0.0016]  [1.0e-24 1.28e-5]      [1.0e-24 1.71799e-24]
 [1.0e-27 0.2]  [1.0e-27 0.0016]  [1.0e-27 1.28e-5]      [1.0e-27 1.71799e-24]
 [1.0e-30 0.2]  [1.0e-30 0.0016]  [1.0e-30 1.28e-5]      [1.0e-30 1.71799e-24]
 [1.0e-33 0.2]  [1.0e-33 0.0016]  [1.0e-33 1.28e-5]   …  [1.0e-33 1.71799e-24]
 [1.0e-36 0.2]  [1.0e-36 0.0016]  [1.0e-36 1.28e-5]      [1.0e-36 1.71799e-24]
```

Рисунок 23. Код и результат Задания 3.11

3.12 вектор с элементами $\frac{2^i}{i}$, i = 1, 2, ... , M, M = 25;

```
M = 25
vector_2 = [(2^i)/i for i=1:1:M]
print(vector_2)
```

```
[2.0, 2.0, 2.6666666666666665, 4.0, 6.4, 10.666666666666666, 18.285714285714285, 32.0, 56.888888888888886, 10
2.4, 186.1818181818182, 341.3333333333333, 630.1538461538462, 1170.2857142857142, 2184.5333333333333, 4096.0,
7710.117647058823, 14563.555555555555, 27594.105263157893, 52428.8, 99864.38095238095, 190650.18181818182, 364
722.0869565217, 699050.6666666666, 1.34217728e6]
```

Рисунок 24. Код и результат Задания 3.12

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

```
N = 30
vector_3 = [string("fn", i) for i=1:1:N]
print(vector_3)
```

```
["fn1", "fn2", "fn3", "fn4", "fn5", "fn6", "fn7", "fn8", "fn9", "fn10", "fn11", "fn12", "fn13", "fn14", "fn1
5", "fn16", "fn17", "fn18", "fn19", "fn20", "fn21", "fn22", "fn23", "fn24", "fn25", "fn26", "fn27", "fn28", "f
n29", "fn30"]
```

Рисунок 25. Код и результат Задания 3.13

3.14 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:

```
n = 250
x = rand(0:999, n)
y = rand(0:999, n)

# проверим длину выборки x
println(length(x))
println(x)
println(y)
```

```
250
[968, 289, 190, 332, 125, 39, 774, 396, 618, 328, 675, 721, 194, 633, 9, 290, 259, 895, 839, 555, 385, 209, 23
1, 762, 662, 166, 532, 588, 289, 47, 341, 993, 150, 795, 364, 497, 649, 289, 855, 31, 787, 37, 571, 918, 127,
45, 934, 536, 7, 360, 17, 779, 914, 511, 590, 668, 251, 327, 501, 709, 724, 2, 434, 85, 791, 985, 495, 794, 9
0, 812, 264, 112, 864, 561, 560, 572, 875, 360, 311, 867, 84, 686, 153, 967, 316, 600, 156, 178, 67, 81, 71, 8
37, 7, 523, 960, 970, 938, 674, 363, 205, 833, 787, 734, 350, 406, 178, 655, 431, 898, 317, 647, 22, 929, 8, 1
28, 314, 751, 650, 41, 186, 516, 870, 840, 136, 774, 56, 157, 274, 416, 655, 444, 823, 566, 927, 622, 151, 60
1, 766, 261, 998, 780, 47, 131, 805, 645, 42, 885, 967, 560, 690, 110, 863, 511, 288, 920, 235, 513, 20, 415,
918, 694, 647, 770, 693, 628, 424, 939, 724, 131, 901, 148, 133, 136, 38, 579, 715, 164, 817, 363, 564, 227, 8
60, 346, 362, 673, 297, 206, 457, 110, 552, 9, 731, 313, 709, 247, 882, 24, 233, 436, 277, 563, 813, 662, 956,
214, 513, 358, 188, 102, 404, 644, 473, 815, 209, 602, 519, 388, 987, 432, 409, 169, 470, 25, 829, 694, 101, 4
86, 645, 653, 977, 507, 536, 183, 536, 376, 236, 590, 182, 969, 936, 287, 612, 176, 499, 101, 945, 633, 941, 5
71, 465]
[691, 273, 344, 358, 540, 178, 232, 422, 464, 944, 982, 668, 248, 826, 13, 306, 511, 47, 506, 237, 351, 31, 41
7, 494, 737, 186, 312, 675, 313, 586, 722, 543, 573, 697, 670, 874, 520, 950, 975, 446, 569, 876, 628, 228, 37
2, 802, 543, 486, 108, 715, 256, 92, 354, 832, 958, 752, 440, 764, 268, 14, 996, 774, 232, 586, 678, 63, 978,
904, 736, 503, 221, 233, 83, 209, 689, 997, 199, 550, 351, 589, 747, 666, 963, 345, 379, 467, 909, 494, 151, 3
40, 321, 537, 736, 144, 805, 720, 892, 895, 401, 337, 910, 323, 278, 188, 32, 229, 799, 879, 497, 68, 901, 59
2, 862, 222, 56, 151, 628, 743, 857, 631, 272, 648, 774, 736, 513, 276, 936, 55, 785, 786, 226, 764, 87, 975,
311, 746, 495, 695, 663, 575, 267, 160, 485, 191, 655, 753, 18, 240, 448, 827, 48, 397, 399, 270, 590, 572, 45
7, 709, 567, 438, 802, 679, 296, 72, 254, 401, 274, 70, 317, 38, 126, 353, 595, 593, 493, 278, 554, 834, 121,
816, 602, 288, 229, 230, 596, 360, 216, 773, 320, 137, 619, 353, 486, 736, 977, 910, 58, 246, 832, 128, 920, 1
6, 623, 379, 937, 45, 557, 134, 767, 359, 270, 147, 786, 578, 628, 48, 597, 879, 306, 649, 881, 305, 306, 140,
500, 245, 184, 48, 336, 617, 486, 506, 546, 541, 679, 568, 4, 615, 526, 204, 425, 970, 601, 488, 671, 88, 719,
920, 384, 259]
```

Рисунок 26. Код и результат Задания 3.14

– сформируйте вектор $(y_2 - x_1, \dots, y_n - x_{n-1})$;

```
vector1 = [y[i]-x[i-1] for i in 2:250]
print(vector1)
```

```
[-695, 55, 168, 208, 53, 193, -352, 68, 326, 654, -7, -473, 632, -620, 297, 221, -212, -389, -602, -204, -354,
208, 263, -25, -476, 146, 143, -275, 297, 675, 202, -420, 547, -125, 510, 23, 301, 686, -409, 538, 89, 591, -3
43, -546, 675, 498, -448, -428, 708, -104, 75, -425, -82, 447, 162, -228, 513, -59, -487, 287, 50, 230, 152, 5
93, -728, -7, 409, -58, 413, -591, -31, -29, -655, 128, 437, -373, -325, -9, 278, -120, 582, 277, 192, -588, 1
51, 309, 338, -27, 273, 240, 466, -101, 137, 282, -240, -78, -43, -273, -26, 705, -510, -509, -546, -318, -17
7, 621, 224, 66, -830, 584, -55, 840, -707, 48, 23, 314, -8, 207, 590, 86, 132, -96, -104, 377, -498, 880, -10
2, 511, 370, -429, 320, -736, 409, -616, 124, 344, 94, -103, 314, -731, -620, 438, 60, -150, 108, -24, -645, -
519, 267, -642, 287, -464, -241, 302, -348, 222, 196, 547, 23, -116, -15, -351, -698, -439, -227, -150, -869,
-407, -93, -775, 205, 462, 457, 455, -301, -161, 670, -696, 453, 38, 61, -631, -116, 234, -313, -81, 567, -13
7, 27, 67, 344, -245, 423, 268, 663, -824, 222, 599, -308, 643, -547, -190, -283, -19, -169, 44, -224, 579, 25
7, -134, -497, 313, -237, 419, -554, 78, 491, -681, 217, 472, 136, -164, 115, -329, -449, 83, -438, -309, -36,
-491, -1, 10, 358, 143, 192, -232, 25, 344, -765, -511, 683, -11, 312, 172, -13, -226, 287, -557, -312]
```

Рисунок 27. Код и результат Задания 3.14-1

– сформируйте вектор $(x_1 + 2x_2 - x_3, x_2 + 2x_3 - x_4, \dots, x_{n-2} + 2x_{n-1} - x_n)$;

```
vector2 = [x[i]+2*x[i+1]-x[i+2] for i in 1:248]
print(vector2)
```

```
[1356, 337, 729, 543, -571, 1191, 948, 1304, 599, 957, 1923, 476, 1451, 361, 330, -87, 1210, 2018, 1564, 1116,
572, -91, 1093, 1920, 462, 642, 1419, 1119, 42, -264, 2177, 498, 1376, 1026, 709, 1506, 372, 1968, 130, 1568,
290, 261, 2280, 1127, -717, 1377, 1999, 190, 710, -385, 661, 2096, 1346, 1023, 1675, 843, 404, 620, 1195, 215
5, 294, 785, -187, 682, 2266, 1181, 1993, 162, 1450, 1228, -376, 1279, 1426, 1109, 829, 1962, 1284, 115, 1961,
349, 1303, 25, 1771, 999, 1360, 734, 445, 231, 158, -614, 1738, 328, 93, 1473, 1962, 2172, 1923, 1195, -60, 10
84, 1673, 1905, 1028, 984, 107, 1057, 619, 1910, 885, 1589, -238, 1872, 817, -50, 5, 1166, 2010, 546, -103, 34
8, 1416, 2414, 338, 1628, 729, 96, 289, 451, 1282, 720, 1524, 1028, 1798, 2020, 323, 587, 1872, 290, 1477, 251
1, 743, -496, 1096, 2053, -156, 845, 2259, 1397, 1830, 47, 1325, 1597, 167, 1893, 877, 1241, 138, -68, 1557, 1
659, 1218, 1494, 1528, 1525, 537, 1578, 2256, 85, 1785, 1064, 278, 367, -367, 481, 1845, 226, 1435, 979, 1264,
158, 1601, 1190, 397, 1411, 1061, 252, 1010, 125, 1205, -161, 1158, 648, 1484, 321, 1987, 697, 54, 828, 427, 5
90, 1527, 1181, 2360, 871, 882, 1041, 632, -12, 266, 1219, 775, 1894, 631, 894, 1252, 308, 1930, 1442, 1081, 2
77, 1084, -309, 989, 2116, 410, 428, 1123, 974, 2100, 1455, 1396, 366, 879, 1052, 258, 1234, -15, 1184, 2554,
898, 1335, 465, 1073, -244, 1358, 1270, 1944, 1618]
```

Рисунок 28. Код и результат Задания 3.14-2

– сформируйте вектор $(\frac{\sin(y1)}{\cos(x2)}, \frac{\sin(y2)}{\cos(x3)}, \dots, \frac{\sin(y_{n-1})}{\cos(x_n)})$;

```
vector3 = [sin(y[i-1])/cos(x[i]) for i in 2:250]
print(vector3)
```

```
[-0.14987031767091288, 4.723495618045471, -1.8766525922911392, -0.17911332486188286, -1.2998390818350767, 2.23
98452986040747, -0.4657760114017271, -1.365598206309524, -2.795951083070179, -1.10523371396021, 215.8593160309
3564, 1.2875845370686796, -5.9761760441179, -0.25968056028362785, 0.7470580136090632, -5.28690402508279, -0.94
00275788884965, -0.12595443536130319, 0.4149704307165043, 6.364726598261218, 9.010447972208764, -4.35356218721
0435, -4.533328436710022, 1.0873351049658793, -1.0924073115163213, 1.2558233183571141, 0.9596814881468934, 0.4
282788308443628, 0.9235883616905673, -7.2801246934080055, -0.5547009053056638, 0.6800094419838516, -0.95750419
47052774, -0.46108291753819297, -0.9210884593408833, -2.3047006070808336, -0.9981463714300118, 1.0697297683019
327, 0.9773650175760045, 3.4140983435174443, -0.47455039024468665, 0.6730288301136562, -0.3948794555309918, 4.
185861685596807, 1.8301266775970157, 1.3344574056745482, -1.355366285972636, 1.076513263483843, -3.26699897610
91958, 3.4848653097612763, -1.0058493575259602, 0.7958828458322693, -1.783956720205399, 0.6124430286904642, -
0.4616853434796277, -0.967966249073741, 0.18294796812812122, 6.658442142377563, -1.5209223528554043, 7.2477722
2397991, 0.2760435465489662, 1.026889780611317, 0.46717557381951813, 1.28189865184507, -4.9908647793048475, 0.
8459010255154696, 1.2085110947351982, 1.5674732665659685, 7.511371510211083, 0.3402794856193902, 1.94297988447
81772, -0.49970002510573663, -4.326924556545901, 1.4251368125442228, -0.8593451720776114, 13.52419890377283,
3.1083064027388656, 0.21951835699977565, -0.7591850140651127, 1.4687994800999447, -1.5172245939114883, 0.02983
184002540898, 1.2141693139618708, 2.0399436292848097, -0.9064507433310887, 1.8869448346124151, 1.8392907111130
645, 1.344423281842723, 0.26027234441579455, -2.1044107144542186, 2.277689241993434, 0.2795476435853696, 10.15
3611282028908, -2.037038283960416, -0.9357335783509765, 2.3406262862419163, 1.645964687502694, 2.3813988054188
684, 1.2896704634545364, 0.8457847994671096, 28.137899882328696, 1.2997592136487475, -3.5239908896831427, 0.64
07517693468878, -1.1501503857704276, 14.953102429979982, -1.0434129286179232, -0.6842447516295289, -0.61546447
17200608, -0.9107633699147457, -0.5948833914433511, 1.6028925291153686, -6.414721316120594, -1.254085839063508
1, -0.5282363300979774, -0.2047440473442852, 0.3288127015911227, -1.0127329891704975, -0.7626962470303496, 0.6
238938059436789, -0.9925325094441608, -2.012936640242088, -1.5020036003160486, 1.9463859633611682, -0.93275151
6529979, -0.4454974443995499, 0.2489930283393263, -3.8739312544858535, -17.570814354318728, -1.109783578310981
5, -0.1943626679585035, -0.641519491586298, 0.844077627635988, 0.894595930920931, 0.018044317689175856, 1.7190
077420122627, -1.1488214656674, 0.6708260517387642, -0.2387440880395451, -0.13963772704117025, -0.035641351494
20598, 0.3755938834760111, 1.2739019168869956, -1.0577957243797316, -2.499483026181436, -1.3896342304221247, -
0.9165757317433685, 1.3521663827417119, 2.323416736669702, 0.6904176423281648, 1.2990038715924068, -1.94313049
61507168, -0.03424529894813157, 0.1991682512081372, 0.7130473919897989, -0.3767291420068118, -2.43781814059908
57, -0.8834891424380289, 1.2588227432854895, 1.0111678037759573, -0.7906466807451711, -0.4242913518769728, -2.
313633621249368, 0.26726784956716787, -0.45500473240587785, 0.955256321547709, -4.608742677603981, 1.324681618
7868582, -0.3686601172353329, -0.31494186279993447, 0.6668202660572994, -1.4844707930886656, -0.98992887182167
45, 1.1805916528724003, 0.8036899463889491, 1.2435327301586483, 0.897274491757768, -6.841843982131883, 11.8878
95988348692, -1.0493986218499947, -1.325330964851261, -0.9389612788497758, -0.4376841109949815, -0.80491785181
60502, 6.584536614637229, 4.285267499995181, -6.853956832454241, -0.16757922977903714, -0.7070219624813275, 1.
0344453411180259, 0.19359509062571123, 2.2732013734976784, 1.5019546888330322, -2.030682363381608, -0.05004002
264515844, -2.0587249925319124, 1.1455094219317554, -1.0515704572190794, 0.5813337673871246, -0.90913880207595
73, -0.5978321522122886, 0.4497149626334062, 1.424530850339192, 0.9719378357129027, -1.1909095647055197, 0.859
5013311197482, -0.9167142761998942, 8.720959586062438, -1.4510582422668363, -0.7570878949271923, 0.93067698155
46634, -2.528197034258996, -6.740812896894384, -0.14118279514044682, 0.38963433261279945, -57.733604475061355,
0.11336771763000199, -19.414565030515117, -1.1500901697234578, 1.2099020939369938, 2.9972038915435006, -0.2647
3284651840445, -1.0272139732159085, -1.023559955397154, -0.5244049905619529, 0.07567621459257047, -1.736446362
743474, -0.8537176729921356, -0.14994089676365688, -2.6413888918772406, -2.3133598120012495, -0.28656400899650
53, 1.6956697804321303, 1.0991201868870137, -0.43522902333038227, 0.7218554427568983, -0.7741961316374948, -3.
788466289773376, -0.9953872620965662, -0.4592434268445509, 0.9445724696411707, 0.6849121244655977, 0.937572118
9892088, -0.9741882615830117, 1.183664563263979, -1.1450287536104669, 4.437172448305226, 0.6513388283037566,
0.6643078012427791]
```

Рисунок 29. Код и результат Задания 3.14-3

– вычислите $\sum_{i=1}^{n-1} \frac{e^{-x_{i+1}}}{x_i+10}$;

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

```
0.00018789610052354785
```

Рисунок 30. Код и результат Задания 3.14-4

– выберите элементы вектора y, значения которых больше 600, и выведите на экран; определите индексы этих элементов;

```
for i in 1:250
    if y[i]>600
        println("i = ", i, "  y = ", y[i])
    end
end
```

```
i = 1  y = 691
i = 10  y = 944
i = 11  y = 982
i = 12  y = 668
i = 14  y = 826
i = 25  y = 737
i = 28  y = 675
i = 31  y = 722
i = 34  y = 697
i = 35  y = 670
i = 36  y = 874
i = 38  y = 950
i = 39  y = 975
i = 42  y = 876
i = 43  y = 628
i = 46  y = 802
i = 50  y = 715
i = 54  y = 832
i = 55  y = 958
```

Рисунок 31. Код и результат Задания 3.14-5

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

```
for i in 1:250
    if y[i]>600
        println("i = ", i, "  y = ", y[i], "  x = ", x[i])
    end
end
```

```
i = 1  y = 691  x = 968
i = 10  y = 944  x = 328
i = 11  y = 982  x = 675
i = 12  y = 668  x = 721
i = 14  y = 826  x = 633
i = 25  y = 737  x = 662
i = 28  y = 675  x = 588
i = 31  y = 722  x = 341
i = 34  y = 697  x = 795
i = 35  y = 670  x = 364
i = 36  y = 874  x = 497
i = 38  y = 950  x = 289
i = 39  y = 975  x = 855
i = 42  y = 876  x = 37
i = 43  y = 628  x = 571
i = 46  y = 802  x = 45
i = 50  y = 715  x = 360
i = 54  y = 832  x = 511
i = 55  y = 958  x = 590
```

Рисунок 32. Код и результат Задания 3.14-6

– сформируйте вектор ($|x_1 - \overline{x}|^{\frac{1}{2}}, |x_2 - \overline{x}|^{\frac{1}{2}}, |x_n - \overline{x}|^{\frac{1}{2}}$), где $\overline{x}$ обозначает среднее значение вектора x = ($x1$, $x2$, … , xn);

```
S = sum(x)/length(x)
vector4 = [(abs(x[i]-S))^0.5 for i in 1:250]
```

```
250-element Vector{Float64}:
 21.98635940759634
 13.985706989637672
 17.16391563717324
 12.353137253345809
 18.963122105813696
 21.109239683134017
 17.01176063786462
  9.412757300600076
 11.549891774384728
 12.513992168768526
 13.798550648528272
 15.375304875026055
 17.046993869888027
  ⋮
 22.009089031579656
 21.24617612654098
 14.057026712644463
 11.287160847617969
 17.567014544310027
  3.7947331922020524
 19.585709075752145
 21.45693361130616
 12.181953866272847
 21.36352030916253
  9.295160030897799
  4.427188724235734
```

Рисунок 33. Код и результат Задания 3.14-7

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

```
count_numbers = 0
for i in 1:250
    if y[i] > maximum(y) - 200
        count_numbers += 1
    end
end
print(count_numbers)
```

```
40
```

Рисунок 34. Код и результат Задания 3.14-8

– определите, сколько чётных и нечётных элементов вектора x;

```
even_num = 0
odd_num = 0
for i in 1:1:n
    if mod(x[i], 2) == 0
        even_num += 1
    else
        odd_num += 1
    end
end
println("Количество четных чисел = ", even_num)
print("Количество нечетных чисел = ", odd_num)
```

```
Количество четных чисел = 120
Количество нечетных чисел = 130
```

Рисунок 35. Код и результат Задания 3.14-9

– определите, сколько элементов вектора x кратны 7;

```
count_seven = 0
for i in 1:250
    if mod(x[i], 7) == 0
        count_seven += 1
    end
end
print("Количество элементов кратных 7 = ", count_seven)
```

```
Количество элементов кратных 7 = 39
```

Рисунок 36. Код и результат Задания 3.14-10

– отсортируйте элементы вектора x в порядке возрастания элементов вектора y;

```
println(sort(y))
mass_new=[]
ind = sortperm(y)
for i in 1:length(x)
    push!(mass_new, x[ind[i]])
end
println(mass_new)
```

```
[4, 13, 14, 16, 18, 31, 32, 38, 45, 47, 48, 48, 48, 55, 56, 58, 63, 68, 70, 72, 83, 87, 88, 92, 108, 121, 126,
128, 134, 137, 140, 144, 147, 151, 151, 160, 178, 184, 186, 188, 191, 199, 204, 209, 216, 221, 222, 226, 228,
229, 229, 230, 232, 232, 233, 237, 240, 245, 246, 248, 254, 256, 259, 267, 268, 270, 270, 272, 273, 274, 276,
278, 278, 288, 296, 305, 306, 306, 306, 311, 312, 313, 317, 320, 321, 323, 336, 337, 340, 344, 345, 351, 351,
353, 353, 354, 358, 359, 360, 372, 379, 379, 384, 397, 399, 401, 401, 417, 422, 425, 438, 440, 446, 448, 457,
464, 467, 485, 486, 486, 486, 488, 493, 494, 494, 495, 497, 500, 503, 506, 506, 511, 513, 520, 526, 537, 540,
541, 543, 543, 546, 550, 554, 557, 567, 568, 569, 572, 573, 575, 578, 586, 586, 589, 590, 592, 593, 595, 596,
597, 601, 602, 615, 617, 619, 623, 628, 628, 628, 631, 648, 649, 655, 663, 666, 668, 670, 671, 675, 678, 679,
679, 689, 691, 695, 697, 709, 715, 719, 720, 722, 736, 736, 736, 736, 737, 743, 746, 747, 752, 753, 764, 764,
767, 773, 774, 774, 785, 786, 786, 799, 802, 802, 805, 816, 826, 827, 832, 832, 834, 857, 862, 874, 876, 879,
879, 881, 892, 895, 901, 904, 909, 910, 910, 920, 920, 936, 937, 944, 950, 958, 963, 970, 975, 975, 977, 978,
982, 996, 997]
Any[590, 9, 709, 813, 885, 209, 406, 901, 513, 895, 110, 519, 645, 274, 128, 24, 985, 317, 724, 693, 864, 566,
945, 779, 7, 363, 148, 277, 188, 552, 829, 523, 473, 67, 314, 47, 39, 486, 166, 350, 805, 875, 936, 561, 206,
264, 8, 444, 918, 178, 346, 362, 774, 434, 112, 555, 967, 101, 233, 194, 628, 17, 465, 780, 501, 288, 644, 51
6, 289, 939, 56, 734, 715, 860, 770, 470, 290, 432, 25, 622, 532, 289, 131, 110, 71, 787, 653, 205, 81, 190, 9
67, 385, 311, 133, 731, 914, 332, 404, 297, 127, 316, 956, 571, 863, 511, 363, 424, 231, 396, 287, 918, 251, 3
1, 560, 513, 618, 600, 131, 536, 313, 507, 499, 579, 762, 178, 601, 898, 694, 812, 839, 536, 259, 774, 649, 96
9, 837, 125, 536, 993, 934, 183, 360, 164, 358, 415, 236, 787, 235, 150, 998, 209, 47, 85, 867, 920, 22, 38, 1
36, 673, 388, 176, 227, 182, 977, 9, 662, 571, 751, 602, 186, 870, 409, 645, 261, 686, 721, 364, 101, 588, 79
1, 647, 376, 560, 968, 766, 795, 20, 360, 633, 970, 341, 90, 7, 136, 709, 662, 650, 151, 84, 668, 42, 327, 82
3, 102, 457, 2, 840, 416, 655, 815, 655, 45, 694, 960, 564, 633, 690, 511, 436, 817, 41, 929, 497, 37, 431, 98
7, 169, 938, 674, 647, 794, 156, 833, 882, 563, 941, 157, 214, 328, 289, 590, 153, 612, 855, 927, 247, 495, 67
5, 724, 572]
```

Рисунок 37. Код и результат Задания 3.14-11

– выведите элементы вектора x, которые входят в десятку наибольших (top-10)?

```
x_sort = sort(x, rev=true)
print(x_sort[1:10])
```

```
[998, 993, 987, 985, 977, 970, 969, 968, 967, 967]
```

Рисунок 38. Код и результат Задания 3.14-12

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

```
x_unique = unique(x)
print(x_unique)
length(x_unique)
```

```
[968, 289, 190, 332, 125, 39, 774, 396, 618, 328, 675, 721, 194, 633, 9, 290, 259, 895, 839, 555, 385, 209, 23
1, 762, 662, 166, 532, 588, 47, 341, 993, 150, 795, 364, 497, 649, 855, 31, 787, 37, 571, 918, 127, 45, 934, 5
36, 7, 360, 17, 779, 914, 511, 590, 668, 251, 327, 501, 709, 724, 2, 434, 85, 791, 985, 495, 794, 90, 812, 26
4, 112, 864, 561, 560, 572, 875, 311, 867, 84, 686, 153, 967, 316, 600, 156, 178, 67, 81, 71, 837, 523, 960, 9
70, 938, 674, 363, 205, 833, 734, 350, 406, 655, 431, 898, 317, 647, 22, 929, 8, 128, 314, 751, 650, 41, 186,
516, 870, 840, 136, 56, 157, 274, 416, 444, 823, 566, 927, 622, 151, 601, 766, 261, 998, 780, 131, 805, 645, 4
2, 885, 690, 110, 863, 288, 920, 235, 513, 20, 415, 694, 770, 693, 628, 424, 939, 901, 148, 133, 38, 579, 715,
164, 817, 564, 227, 860, 346, 362, 673, 297, 206, 457, 552, 731, 313, 247, 882, 24, 233, 436, 277, 563, 813, 9
56, 214, 358, 188, 102, 404, 644, 473, 815, 602, 519, 388, 987, 432, 409, 169, 470, 25, 829, 101, 486, 653, 97
7, 507, 183, 376, 236, 182, 969, 936, 287, 612, 176, 499, 945, 941, 465]
```

```
218
```

Рисунок 39. Код и результат Задания 3.14-13

4. Создайте массив squares, в котором будут храниться квадраты всех целых чисел от 1 до 100.

```
# зададим в цикле получение квадратов всех целых чисел от 1 до 100

squares = [i^2 for i=1:100]
print(squares)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361, 400, 441, 484, 529, 576, 6
25, 676, 729, 784, 841, 900, 961, 1024, 1089, 1156, 1225, 1296, 1369, 1444, 1521, 1600, 1681, 1764, 1849, 193
6, 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, 65
61, 6724, 6889, 7056, 7225, 7396, 7569, 7744, 7921, 8100, 8281, 8464, 8649, 8836, 9025, 9216, 9409, 9604, 980
1, 10000]
```

Рисунок 40. Код и результат Задания 4

5. Подключите пакет Primes (функции для вычисления простых чисел). Сгенерируйте массив myprimes, в котором будут храниться первые 168 простых чисел. Определите 89-е наименьшее простое число. Получите срез массива с 89-го до 99-го элемента включительно, содержащий наименьшие простые числа.

```
import Pkg
Pkg.add("Primes")
```

```
    Updating registry at `C:\Users\GanZL\.julia\registries\General.toml`
   Resolving package versions...
   Installed IntegerMathUtils ─ v0.1.2
   Installed Primes ──────────── v0.5.6
    Updating `C:\Users\GanZL\.julia\environments\v1.11\Project.toml`
  [27ebfcd6] + Primes v0.5.6
    Updating `C:\Users\GanZL\.julia\environments\v1.11\Manifest.toml`
  [18e54dd8] + IntegerMathUtils v0.1.2
  [27ebfcd6] + Primes v0.5.6
Precompiling project...
   1205.9 ms  ✓ IntegerMathUtils
    511.7 ms  ✓ Primes
  2 dependencies successfully precompiled in 2 seconds. 43 already precompiled.
```

Рисунок 41. Код и результат Задания 5-1

```
# использую пакет Primes и получим первые 168 простых чисел
using Primes
n = 1000
myprimes = primes(n)
print(myprimes)
```

```
[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, 10
7, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 22
9, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 35
9, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 49
1, 499, 503, 509, 521, 523, 541, 547, 557, 563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 64
1, 643, 647, 653, 659, 661, 673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773, 78
7, 797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881, 883, 887, 907, 911, 919, 929, 937, 94
1, 947, 953, 967, 971, 977, 983, 991, 997]
```

```
# определим наименьшее 89-е число
println("89-е наименьшее простое число = ", myprimes[89])
print("срез массива с 89-го до 99-го элемента = ", myprimes[89:99])
```

```
89-е наименьшее простое число = 461
срез массива с 89-го до 99-го элемента = [461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523]
```

Рисунок 42. Код и результат Задания 5-2

6. Вычислите следующие выражения:

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

```
sum((i^3)+4*(i^2) for i=10:100)
```

```
26852735
```

Рисунок 43. Код и результат Задания 6-1

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

```
M = 25
sum(((2^i)/i) + ((3^i)/(i^2)) for i=1:M)
```

```
2.1291704368143802e9
```

Рисунок 44. Код и результат Задания 6-2

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

```
S = 1
temp = 1
for i in 2:2:38
    temp *= i/(i+1)
    S += temp
end
print(S)
```

```
6.976346137897618
```

Рисунок 45. Код и результат Задания 6-3

Вывод

Изучила структуры данных, реализованных в Julia. Научилась применять их и операции над ними для решения задач.