Лабораторная работа №2

Структуры данных

Лихтенштейн Алина Алексеевна

2025-09-26

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

1. Используя Jupyter Lab, повторите примеры.

2. Выполните задания для самостоятельной работы.

3 Теоретическое введение

Julia — высокоуровневый свободный язык программирования с динамической типизацией, созданный для математических вычислений.
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.
Некоторые функции (методы), общие для всех структур данных:

- isempty() — проверяет, пуста ли структура данных;
- length() — возвращает длину структуры данных;
- in() — проверяет принадлежность элемента к структуре;
- unique() — возвращает коллекцию уникальных элементов структуры;
- reduce() — свёртывает структуру данных в соответствии с заданным бинарным оператором;
- maximum() (или `minimum()`) — возвращает наибольший (или наименьший) результат вызова функции для каждого элемента структуры данных.

4 Выполнение лабораторной работы

Для начала выполним примеры из раздела про кортежи (рис. [-@fig:001]).

Кортеж (Tuple) — структура данных (контейнер) в виде неизменяемой индексируемой последовательности элементов какого-либо типа.

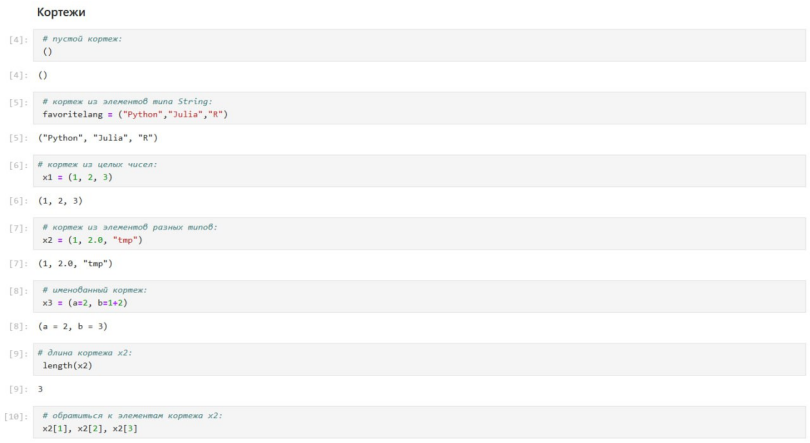


Рисунок 4.1: Примеры использования кортежей

Теперь выполним примеры из раздела про словари (рис. [-@fig:002]).

Словарь — неупорядоченный набор связанных между собой по ключу данных.

Рисунок 4.2: Примеры использования словарей

Выполним примеры из раздела про множества (рис. [-@fig:003]).

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.

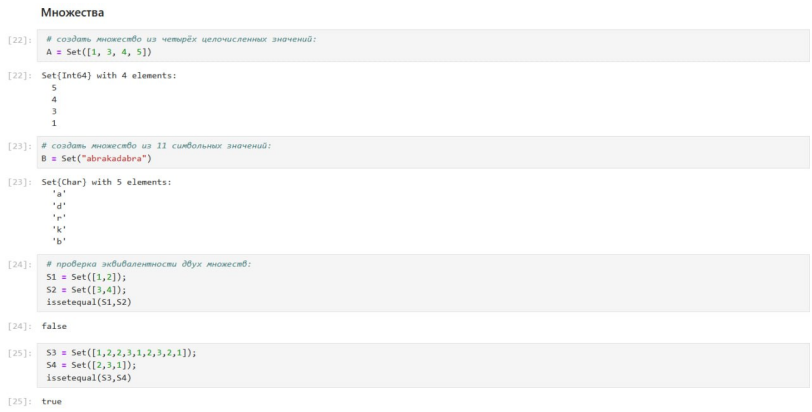
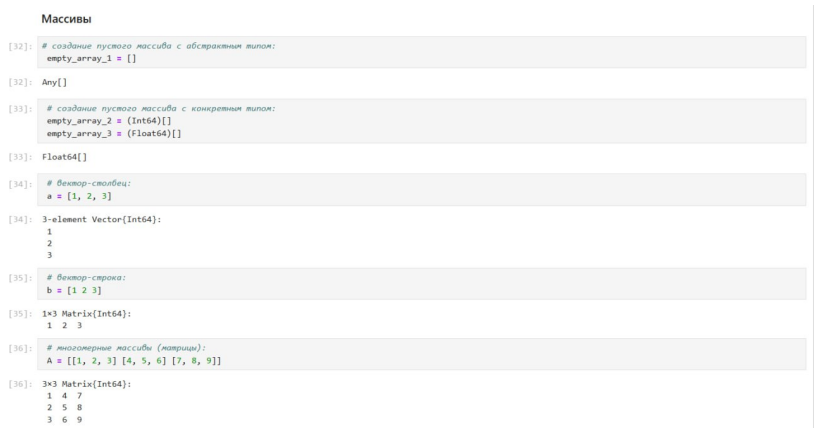


Рисунок 4.3: Примеры использования множеств

Выполним примеры из раздела про массивы (рис. [-@fig:004]–[-@fig:006]).

Массив — коллекция упорядоченных элементов, размещённая в многомерной сетке.

Векторы и матрицы являются частными случаями массивов.



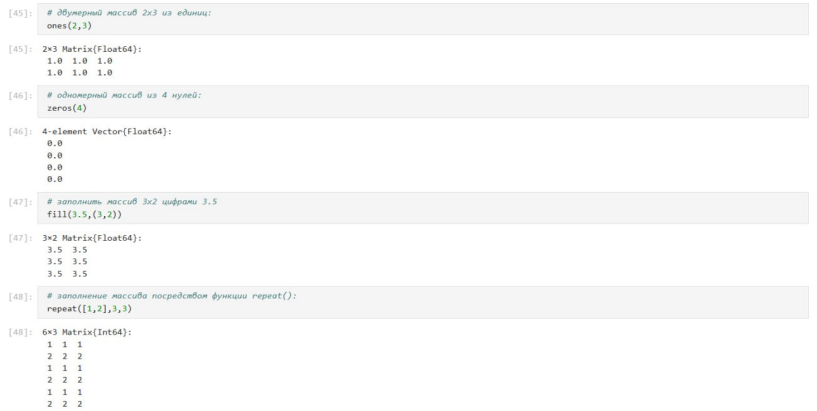
Массивы

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

[32]: Any[]

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

[33]: Float64[]

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

[34]: 3-element Vector{Int64}:
       1
       2
       3

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

[35]: 1×3 Matrix{Int64}:
       1  2  3

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

[36]: 3×3 Matrix{Int64}:
       1  4  7
       2  5  8
       3  6  9
```

Рисунок 4.4: Примеры использования массивов

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

[37]: 3×3 Matrix{Int64}:
       1  2  3
       4  5  6
       7  8  9

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

[38]: 1×8 Matrix{Float64}:
       0.470205  0.121101  0.109701  0.170352  …  0.483343  0.171508  0.0991006

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

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

[40]: 4×3×2 Array{Float64, 3}:
      [:, :, 1] =
       0.581756   0.158493  0.0170796
       0.879408   0.404623  0.244413
       0.944041   0.150032  0.950363
       0.0781636  0.525627  0.608632

      [:, :, 2] =
       0.538072  0.146319  0.547317
       0.32679   0.580525  0.915178
       0.609314  0.128923  0.929443
       0.412019  0.382324  0.656777
```

Рисунок 4.5: Примеры использования массивов



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

[45]: 2×3 Matrix{Float64}:
       1.0  1.0  1.0
       1.0  1.0  1.0

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

[46]: 4-element Vector{Float64}:
       0.0
       0.0
       0.0
       0.0

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

[47]: 3×2 Matrix{Float64}:
       3.5  3.5
       3.5  3.5
       3.5  3.5

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

[48]: 6×3 Matrix{Int64}:
       1  1  1
       2  2  2
       1  1  1
       2  2  2
       1  1  1
       2  2  2
```

Рисунок 4.6: Примеры использования массивов

Теперь перейдем к выполнению заданий.

4.1 Задание №1

Даны множества:

$A = \{0, 3, 4, 9\}, B = \{1, 3, 4, 7\}, C = \{0, 1, 2, 4, 7, 8, 9\}$.

Найдём:

$P = A \cap B \cup A \cap C \cup B \cap C$ (рис. [-@fig:007]).

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

      # Находим пересечения и объединяем их в одно множество
      P = union(intersect(A,B),
                intersect(A,B),
                intersect(A,C),
                intersect(B,C))
      println(P)

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

Рисунок 4.7: Задание №1. Работа с множествами

4.2 Задание №2

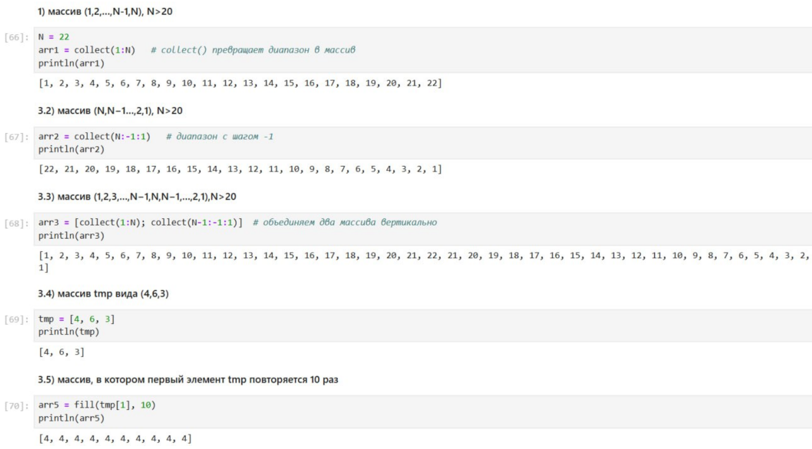
Приведём свои примеры с выполнением операций над множествами элементов разных типов (рис. [-@fig:008]).

```
[65]: S1 = Set([1, "apple", 3.5])      # числа, строка и число с плавающей точкой
      S2 = Set([2, "banana", 3.5])     # числа, строка и число с плавающей точкой
      S3 = Set(["apple", "banana"])    # только строки

      # Пересечение множеств
      intersection1 = intersect(S1, S2)
      println("Пересечение S1 и S2: ", intersection1)

      # Объединение множеств
      union1 = union(S1, S2)
      println("Объединение S1 и S2: ", union1)

      # Разность множеств
      diff1 = setdiff(S1, S2)
      println("Разность S1 и S2: ", diff1)

      # Проверка вхождения элементов
      println("3.5 ∈ S1? ", 3.5 in S1)
      println("\"banana\" ∈ S1? ", "banana" in S1)

      # Добавление элемента в множество
      push!(S1, "orange")
      println("S1 после добавления элемента: ", S1)

      # Удаление последнего элемента множества
      pop!(S1)
      println("S1 после удаления последнего элемента: ", S1)

      # Проверка, является ли одно множество подмножеством другого
      println("S3 ⊆ S2? ", issubset(S3, S2))
```

Рисунок 4.8: Задание №2. Примеры операций над множествами элементов разных типов

4.3 Задание №3

Создадим массивы разными способами, используя циклы (рис. [-@fig:009]–[-@fig:021]).



1) массив (1,2,...,N-1,N), N>20

```
[66]: N = 22
      arr1 = collect(1:N)   # collect() превращает диапазон в массив
      println(arr1)
```

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

3.2) массив (N,N−1...,2,1), N>20

```
[67]: arr2 = collect(N:-1:1)   # диапазон с шагом -1
      println(arr2)
```

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

3.3) массив (1,2,3,...,N−1,N,N−1,...,2,1),N>20

```
[68]: arr3 = [collect(1:N); collect(N-1:-1:1)]  # объединяем два массива вертикально
      println(arr3)
```

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

3.4) массив tmp вида (4,6,3)

```
[69]: tmp = [4, 6, 3]
      println(tmp)
```

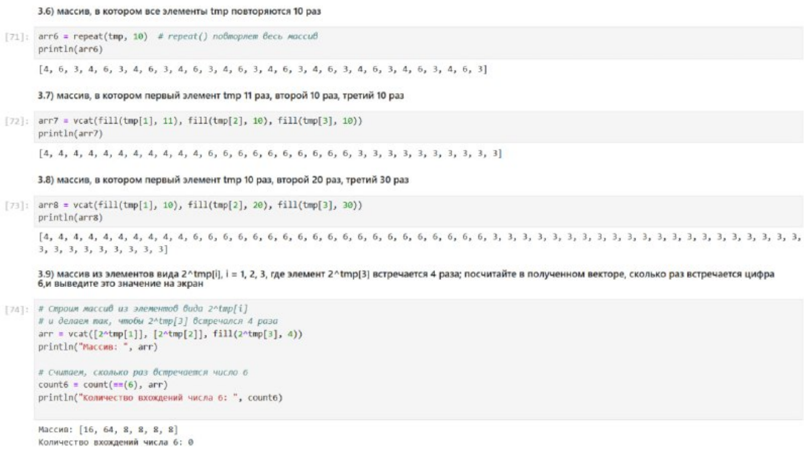
```
[4, 6, 3]
```

3.5) массив, в котором первый элемент tmp повторяется 10 раз

```
[70]: arr5 = fill(tmp[1], 10)
      println(arr5)
```

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

Рисунок 4.9: Задание №3. Работа с массивами



3.6) массив, в котором все элементы tmp повторяются 10 раз

```
[71]: arr6 = repeat(tmp, 10)  # repeat() повторяет весь массив
      println(arr6)
```

```
[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]
```

3.7) массив, в котором первый элемент tmp 11 раз, второй 10 раз, третий 10 раз

```
[72]: arr7 = vcat(fill(tmp[1], 11), fill(tmp[2], 10), fill(tmp[3], 10))
      println(arr7)
```

```
[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]
```

3.8) массив, в котором первый элемент tmp 10 раз, второй 20 раз, третий 30 раз

```
[73]: arr8 = vcat(fill(tmp[1], 10), fill(tmp[2], 20), fill(tmp[3], 30))
      println(arr8)
```

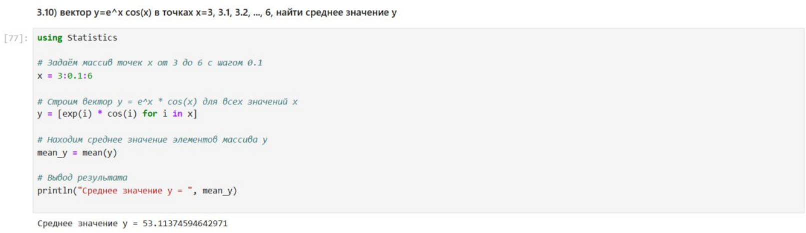
```
[4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3, 3]
```

3.9) 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

```
[74]: # Строим массив из элементов вида 2^tmp[i]
      # и делаем так, чтобы 2^tmp[3] встречался 4 раза
      arr = vcat([2^tmp[1]], [2^tmp[2]], fill(2^tmp[3], 4))
      println("Массив: ", arr)

      # Считаем, сколько раз встречается число 6
      count6 = count(==(6), arr)
      println("Количество вхождений числа 6: ", count6)
```

```
Массив: [16, 64, 8, 8, 8, 8]
Количество вхождений числа 6: 0
```

Рисунок 4.10: Задание №3. Работа с массивами



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

```
[77]: using Statistics

      # Задаём массив точек x от 3 до 6 с шагом 0.1
      x = 3:0.1:6

      # Строим вектор y = e^x * cos(x) для всех значений x
      y = [exp(i) * cos(i) for i in x]

      # Находим среднее значение элементов массива y
      mean_y = mean(y)

      # Вывод результата
      println("Среднее значение y = ", mean_y)
```

```
Среднее значение y = 53.11374594642971
```

Рисунок 4.11: Задание №3. Работа с векторами

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

[78]:
```
# Задаём параметры
x = 0.1
y = 0.2

# Степени i и j
i_vals = 3:3:36    # 3,6,9,...,36
j_vals = 1:3:34    # 1,4,7,...,34

# Строим векторы
vec_x = [x^i for i in i_vals]   # (x^i)
vec_y = [y^j for j in j_vals]   # (y^j)

# Объединяем их в один вектор
vec = vcat(vec_x, vec_y)

# Вывод результата
println("Вектор: ", vec)
```

```
Вектор: [0.001000000000000002, 1.0000000000000004e-6, 1.0000000000000005e-9, 1.0000000000000006e-12, 1.0000000000000009e-15, 1.000000000000001e-18, 1.0000000000000012e-21, 1.0000000000000014e-24, 1.0000000000000015e-27, 1.0000000000000017e-30, 1.0000000000000018e-33, 1.000000000000002e-36, 0.2, 0.0016000000000000003, 1.2800000000000005e-5, 1.0240000000000006e-7, 8.192000000000005e-10, 6.553600000000055e-12, 5.242880000000056e-14, 4.194304000000005e-16, 3.3554432000000048e-18, 2.684354560000004e-20, 2.1474836480000035e-22, 1.7179869184000036-24]
```

Рисунок 4.12: Задание №3. Работа с векторами

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

[79]:
```
# Задаём M
M = 25

# Строим вектор по формуле 2^i / i
y = [2^i / i for i in 1:M]

# Выводим результат
println("Вектор y: ", y)
```

```
Вектор y: [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]
```

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

[80]:
```
N = 30

# Формируем вектор строк
fn_vector = ["fn$(i)" for i in 1:N]

println("Вектор fn_vector: ", fn_vector)
```

```
Вектор fn_vector: ["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"]
```

Рисунок 4.13: Задание №3. Работа с векторами

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:

- сформируйте вектор (y2 − x1 ,..., yn − xn−1)

[81]:
```
using Random

# Длина векторов
n = 250

# Генерация случайных целочисленных векторов x и y из диапазона 0:999
x = rand(0:999, n)
y = rand(0:999, n)

# Формируем новый вектор (y2 - x1, ..., yn - x_{n-1})
z = [y[i+1] - x[i] for i in 1:(n-1)]

# Вывод результата
println("Длина вектора z = ", length(z))
println("Пример первых 10 элементов z: ", z[1:10])
```

```
Длина вектора z = 249
Пример первых 10 элементов z: [608, -602, -82, 457, 73, 411, -198, 414, 651, -497]
```

- сформируйте вектор(x1 + 2x2 −x3, x2 + 2x3 − x4,..., xn−2 + 2xn−1 − xn)

[82]:
```
# Формирование нового вектора (x1 + 2x2 - x3, ..., x_{n-2} + 2x_{n-1} - xn)
z = [x[i] + 2*x[i+1] - x[i+2] for i in 1:(n-2)]

# Вывод результата
println("Длина вектора z = ", length(z))
println("Первые 10 элементов z: ", z[1:10])
```

```
Длина вектора z = 248
Первые 10 элементов z: [1408, 1339, 902, -129, 514, 702, 716, -557, 1156, 2103]
```

Рисунок 4.14: Задание №3. Работа с векторами

- сформируйте вектор (sin(y1)/cos(x2), sin(y2)/cos(x3),...,sin(yn-1)/cos(xn))

```
[83]: # Формирование результирующего вектора
result = [sin(y[i]) / cos(x[i+1]) for i in 1:n-1]

println("Длина результата: ", length(result))
println("Первые 10 значений: ", result[1:10])
```

```
Длина результата: 249
Первые 10 значений: [0.058314237725356245, 1.2919293122304518, 0.4620920945755471, -0.49119286759821895, 7.816468564043453, 31.293182524004845, -0.780266
8223463821, 6.097821294378157, -1.9572582430984606, 1.578836222228221]
```

- вычислите sum(exp^(-x[i+1]) / (x[i] + 10) for i = 1 to n-1)

```
[84]: # Вычисление суммы
S = sum([exp(-x[i+1]) / (x[i] + 10) for i in 1:n-1])

println("Сумма S = ", S)
```

```
Сумма S = 1.3704434342112695e-5
```

Рисунок 4.15: Задание №3. Работа с векторами

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

```
[85]: # Выбор элементов, больших 600
selected_values = y[y .> 600]

# Индексы этих элементов
indices = findall(y .> 600)

# Вывод результатов
println("Элементы больше 600: ", selected_values)
println("Их индексы: ", indices)
```

```
Элементы больше 600: [820, 994, 699, 802, 659, 703, 658, 821, 983, 624, 611, 916, 705, 951, 702, 743, 960, 651, 914, 606, 950, 950, 700, 885, 867, 993, 6
94, 813, 926, 653, 824, 678, 703, 879, 849, 704, 994, 780, 948, 858, 645, 658, 771, 701, 647, 955, 756, 972, 667, 943, 665, 720, 831, 916, 745, 608, 834,
702, 883, 790, 849, 851, 833, 712, 780, 933, 799, 749, 894, 741, 626, 697, 630, 912, 817, 927, 783, 815, 884, 617, 642, 825, 793, 604, 753, 901, 817, 87
6, 906, 640, 725, 969, 992, 902, 958, 848, 752, 687, 995, 648, 885, 611, 737]
Их индексы: [1, 2, 5, 7, 9, 10, 12, 13, 14, 16, 18, 21, 22, 25, 29, 30, 36, 37, 42, 46, 47, 48, 49, 50, 52, 54, 57, 58, 65, 66, 68, 69, 71, 75, 78, 79, 8
2, 83, 86, 87, 88, 95, 96, 101, 102, 103, 105, 106, 109, 111, 112, 114, 115, 119, 120, 121, 124, 125, 126, 128, 129, 133, 137, 144, 146, 147, 148, 154, 1
56, 157, 159, 165, 166, 171, 175, 176, 178, 179, 183, 186, 188, 194, 196, 197, 201, 202, 206, 208, 209, 211, 213, 214, 217, 223, 224, 225, 230, 231, 235,
236, 244, 247, 248]
```

Рисунок 4.16: Задание №3. Работа с векторами

- определите значения вектора x,соответствующиесозначениям вектора y,значения которых больше 600 (под соответствием понимается расположение на аналогичных индексных позициях)

```
[86]: # Находим индексы элементов y, больших 600
indices = findall(y .> 600)

# Берём элементы x, соответствующие этим индексам
corresponding_x = x[indices]

# Вывод результатов
println("Индексы элементов y > 600: ", indices)
println("Элементы x на этих позициях: ", corresponding_x)
```

```
Индексы элементов y > 600: [1, 2, 5, 7, 9, 10, 12, 13, 14, 16, 18, 21, 22, 25, 29, 30, 36, 37, 42, 46, 47, 48, 49, 50, 52, 54, 57, 58, 65, 66, 68, 69, 7
1, 75, 78, 79, 82, 83, 86, 87, 88, 95, 96, 101, 102, 103, 105, 106, 109, 111, 112, 114, 115, 119, 120, 121, 124, 125, 126, 128, 129, 133, 137, 144, 146,
147, 148, 154, 156, 157, 159, 165, 166, 171, 175, 176, 178, 179, 183, 186, 188, 194, 196, 197, 201, 202, 206, 208, 209, 211, 213, 214, 217, 223, 224, 22
5, 230, 231, 235, 236, 244, 247, 248]
Элементы x на этих позициях: [386, 725, 10, 278, 52, 906, 219, 392, 917, 170, 445, 508, 422, 667, 655, 951, 955, 178, 66, 958, 439, 176, 332, 246, 632, 7
19, 747, 685, 322, 438, 457, 796, 361, 433, 700, 665, 41, 736, 542, 524, 438, 506, 960, 114, 38, 81, 598, 297, 985, 235, 541, 246, 587, 657, 61, 925, 11
9, 735, 502, 945, 430, 977, 426, 991, 101, 30, 115, 446, 926, 139, 627, 279, 658, 333, 956, 357, 493, 20, 886, 23, 964, 714, 574, 373, 720, 341, 568, 52
9, 874, 822, 847, 579, 231, 725, 65, 390, 539, 629, 779, 132, 240, 821, 797]
```

Рисунок 4.17: Задание №3. Работа с векторами

- сформируйте вектор (|x1 - X̄|^1/2, |x2 - X̄|^1/2, ..., |xn - X̄|^1/2), где X̄ обозначает среднее значение вектора x = (x1, x2, ..., xn)

```
[87]: # Среднее значение вектора x
x_mean = mean(x)

# Формирование нового вектора (|xi - X̄|^(1/2))
result = [sqrt(abs(x[i] - x_mean)) for i in 1:n]

# Вывод результатов
println("Среднее значение X̄ = ", x_mean)
println("Первые 10 элементов нового вектора: ", result[1:10])
```

```
Среднее значение X̄ = 506.648
Первые 10 элементов нового вектора: [10.983988346679908, 14.7767384764027, 8.86837076356193, 16.268005409391773, 22.285600732311437, 10.753976008900151,
15.121111070288453, 16.175537085364432, 21.322476404020243, 19.9837934336802]
```

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

```
[88]: # Максимальное значение вектора y
y_max = maximum(y)

# Логический вектор: true, если элемент отличается от максимума не более чем на 200
mask = (y_max .- y) .<= 200

# Подсчёт количества таких элементов
count_close = count(mask)

# Вывод результата
println("Максимальное значение вектора y = ", y_max)
println("Количество элементов, которые отличаются от max не более чем на 200: ", count_close)
```

```
Максимальное значение вектора y = 995
Количество элементов, которые отличаются от max не более чем на 200: 52
```

Рисунок 4.18: Задание №3. Работа с векторами

- определите, сколько чётных и нечётных элементов вектора x

[89]:
```
# Подсчёт количества чётных и нечётных элементов
count_even = count(iseven, x)   # сколько элементов чётные
count_odd  = count(isodd, x)    # сколько элементов нечётные

# Вывод результатов
println("Количество чётных элементов: ", count_even)
println("Количество нечётных элементов: ", count_odd)
```

```
Количество чётных элементов: 118
Количество нечётных элементов: 132
```

– определите, сколько элементов вектора x кратны 7

[90]:
```
# Подсчёт количества элементов, кратных 7
count_mult7 = count(xi -> xi % 7 == 0, x)

# Вывод результата
println("Количество элементов вектора x, кратных 7: ", count_mult7)
```

```
Количество элементов вектора x, кратных 7: 35
```

Рисунок 4.19: Задание №3. Работа с векторами

– отсортируйте элементы вектора x в порядке возрастания элементов вектора y

[91]:
```
# Сортировка x по возрастанию соответствующих y
x_sorted = x[sortperm(y)]

# Вывод результатов
println("Первые 10 значений y: ", y[1:10])
println("Первые 10 значений x: ", x[1:10])
println("Первые 10 значений отсортированного x: ", x_sorted[1:10])
```

```
Первые 10 значений y: [820, 994, 123, 346, 699, 83, 802, 80, 659, 703]
Первые 10 значений x: [386, 725, 428, 242, 10, 391, 278, 245, 52, 906]
Первые 10 значений отсортированного x: [585, 983, 542, 317, 935, 150, 787, 760, 395, 148]
```

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

[92]:
```
# Сортировка по убыванию и выбор top-10
top10 = sort(x, rev=true)[1:10]

# Вывод результатов
println("Top-10 элементов вектора x: ", top10)
```

```
Top-10 элементов вектора x: [999, 999, 991, 991, 988, 987, 985, 983, 983, 979]
```

Рисунок 4.20: Задание №3. Работа с векторами

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

[93]:
```
# Уникальные (неповторяющиеся) элементы
unique_x = unique(x)

# Вывод результатов
println("Исходная длина вектора x: ", length(x))
println("Длина вектора уникальных элементов: ", length(unique_x))
println("Первые 20 уникальных элементов: ", unique_x[1:20])
```

```
Исходная длина вектора x: 250
Длина вектора уникальных элементов: 226
Первые 20 уникальных элементов: [386, 725, 428, 242, 10, 391, 278, 245, 52, 906, 708, 219, 392, 917, 649, 170, 585, 445, 541, 251]
```

Рисунок 4.21: Задание №3. Работа с векторами

4.4 Задание №4

Создадим массив squares, в котором будут храниться квадраты всех целых чисел от 1 до 100 (рис. [-@fig:022]).

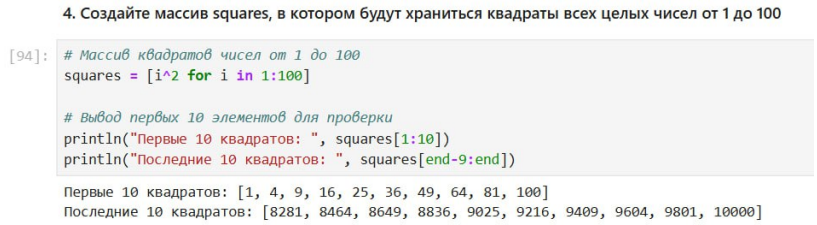


Рисунок 4.22: Задание №4

4.5 Задание №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.

Получим срез массива с 89-го до 99-го элемента включительно, содержащий наименьшие простые числа (рис. [-@fig:023]).

Рисунок 4.23: Задание №5. Работа с пакетом Primes

4.6 Задание №6

Вычислим следующие выражения (рис. [-@fig:024]–[-@fig:025]).

6.1) sum(i^3 + 4*i^2 for i in 10:100)

[97]:
```
# Сумма от i = 10 до 100 для выражения i^3 + 4*i^2
S = sum(i^3 + 4*i^2 for i in 10:100)

println("Сумма S = ", S)
```

```
Сумма S = 26852735
```

6.2) sum(2^i/i + 3^i/i^2 for i in 1:M)

[98]:
```
# Верхняя граница
M = 25

# Вычисление суммы
S = sum(2^i/i + 3^i/i^2 for i in 1:M)

println("Сумма S = ", S)
```

```
Сумма S = 2.1291704368143802e9
```

Рисунок 4.24: Задание №6

6.3) 1 + 2/3 + (2/3 4/5) + (2/3 4/5 6/7) + ... + (2/3 4/5 ... 38/39)

[100]:
```
# Сумма последовательных произведений
S = 0.0  # начальная сумма
prod = 1.0  # текущее произведение

# Проходим по чётным числителям от 2 до 38
for n in 2:2:38
    prod *= n / (n + 1)  # умножаем на дробь n/(n+1)
    S += prod            # добавляем текущее произведение в сумму
end

# Добавляем первый член 1
S += 1.0

println("Сумма S = ", S)
```

```
Сумма S = 6.976346137897619
```

Рисунок 4.25: Задание №6

5 Выводы

В результате выполнения данной лабораторной работы мы изучили несколько структур данных, реализованных в Julia,
и научились применять их и операции над ними для решения задач.

Список литературы

1. JuliaLang [Электронный ресурс]. 2024 JuliaLang.org contributors.
   URL: https://julialang.org/ (дата обращения: 11.10.2024).

2. Julia 1.11 Documentation [Электронный ресурс]. 2024 JuliaLang.org contributors.
   URL: https://docs.julialang.org/en/v1/ (дата обращения: 11.10.2024).