Отчет по лабораторной работе №2

Компьютерный практикум по статистическому анализу данных

Амуничников Антон Игоревич

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 [**julialang**]. 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 [**juliadoc**].

Рассмотрим несколько структур данных, реализованных в Julia. Несколько функций (методов), общих для всех структур данных:

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

4 Выполнение лабораторной работы

Для начала выполним примеры из раздела про кортежи (рис. 4.1). Кортеж (Tuple) – структура данных (контейнер) в виде неизменяемой индексируемой последовательности элементов какого-либо типа (элементы индексируются с единицы).

```
[1]: # пустой кортеж:
     ()

[1]: ()

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

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

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

[3]: (1, 2, 3)

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

[4]: (1, 2.0, "tmp")

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

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

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

[6]: 3

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

[7]: (1, 2.0, "tmp")

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

[8]: 5

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

[9]: (2, 3, 3)

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

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

Рисунок 4.1: Примеры использования кортежей

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

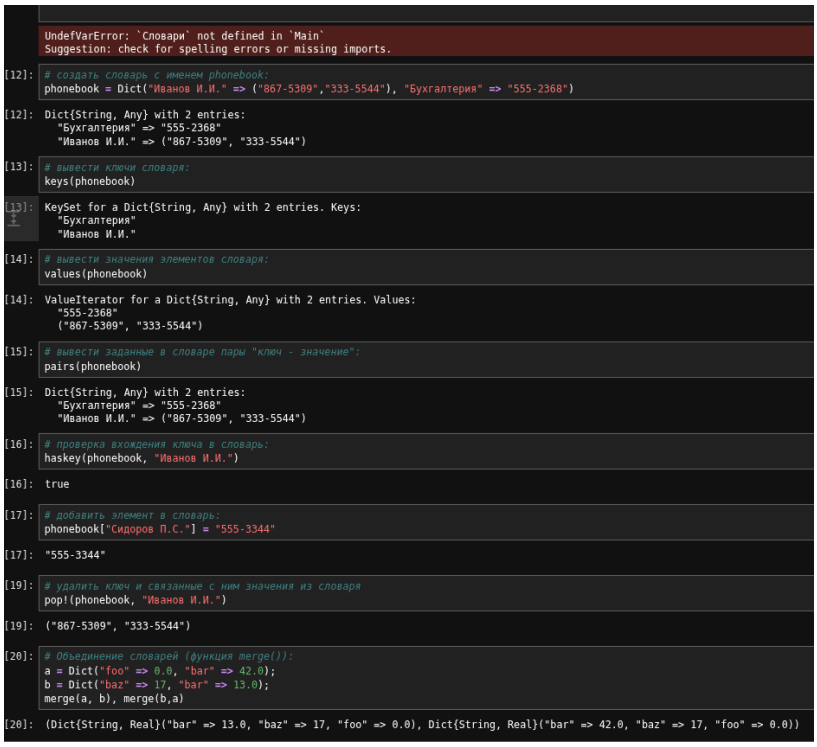

Рисунок 4.2: Примеры использования словарей

Выполним примеры из раздела про множества (рис. 4.3). Множество, как структура данных в Julia, соответствует множеству, как математическому объекту, то есть является неупорядоченной совокупностью элементов какого-либо типа. Возможные операции над множествами: объединение, пересечение, разность; принадлежность элемента множеству.

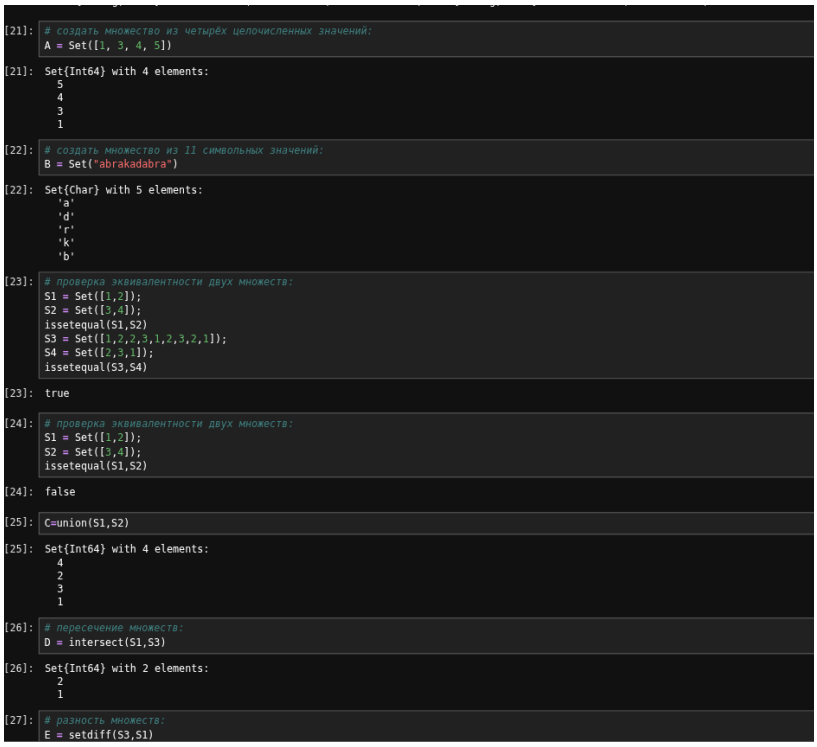

Рисунок 4.3: Примеры использования множеств

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

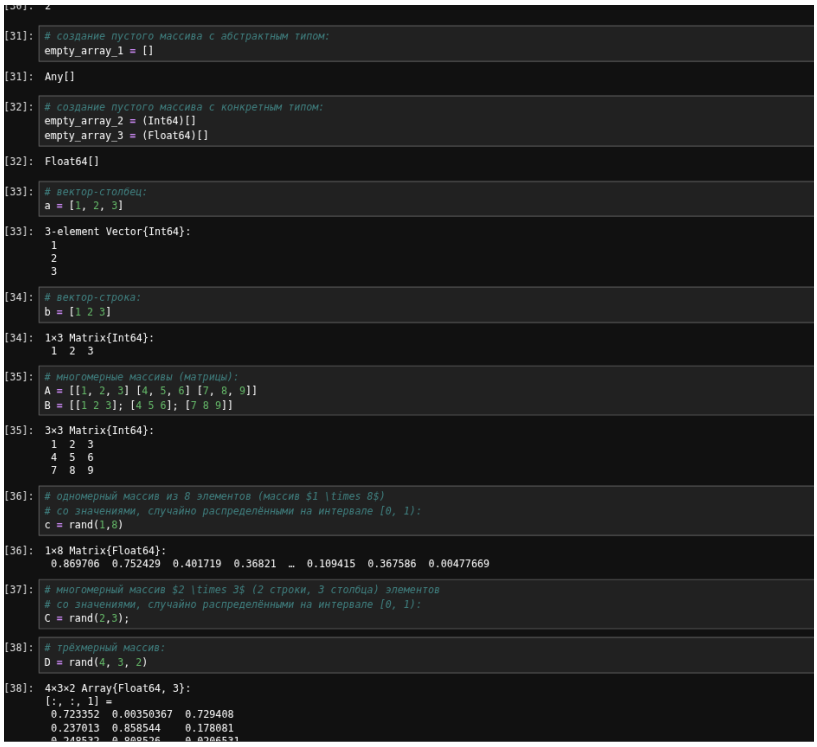

Рисунок 4.4: Примеры использования массивов

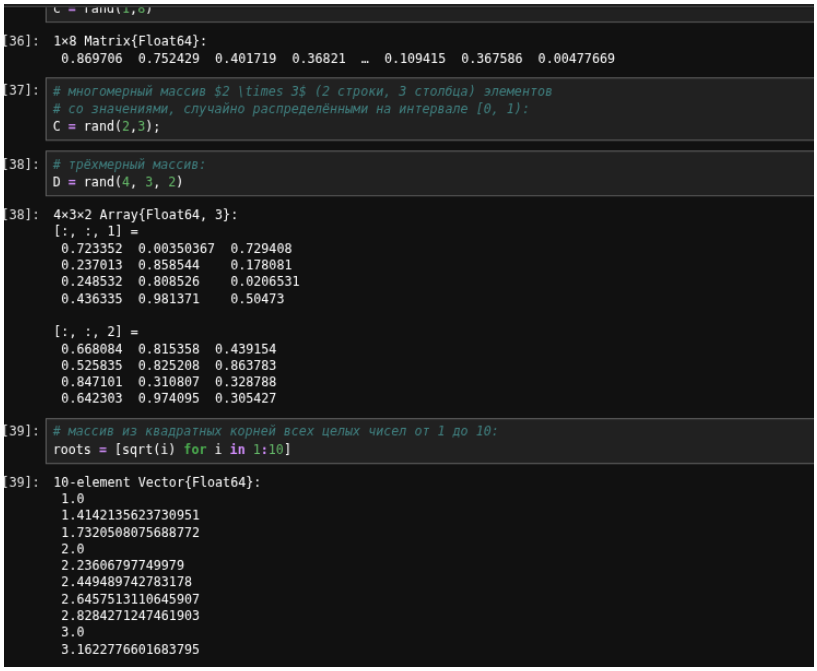

Рисунок 4.5: Примеры использования массивов

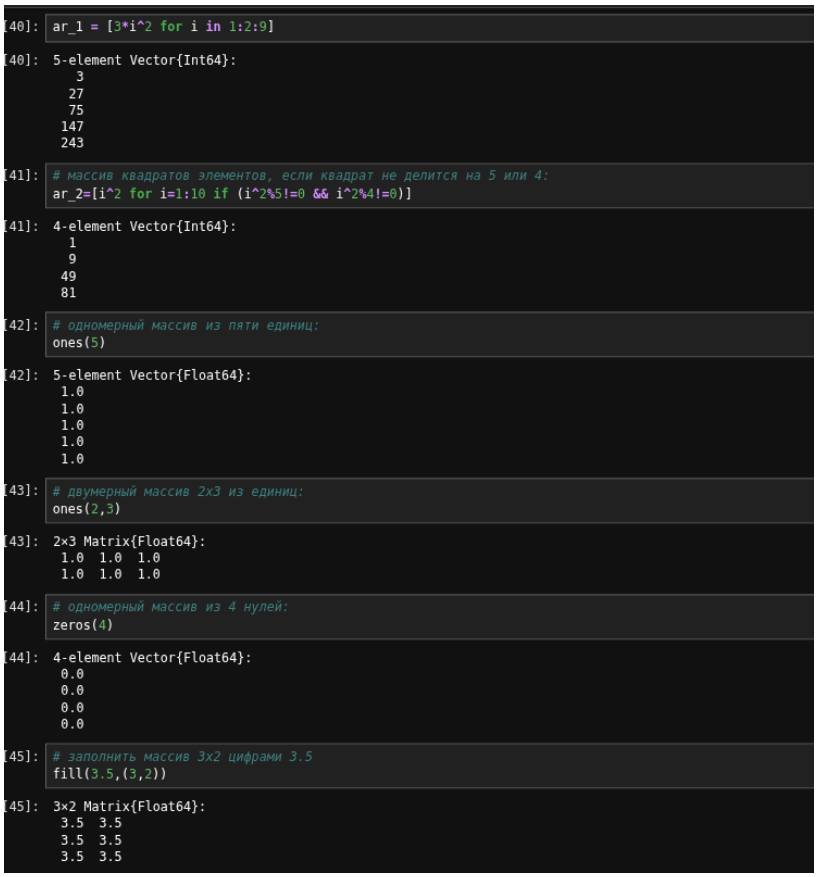

Рисунок 4.6: Примеры использования массивов

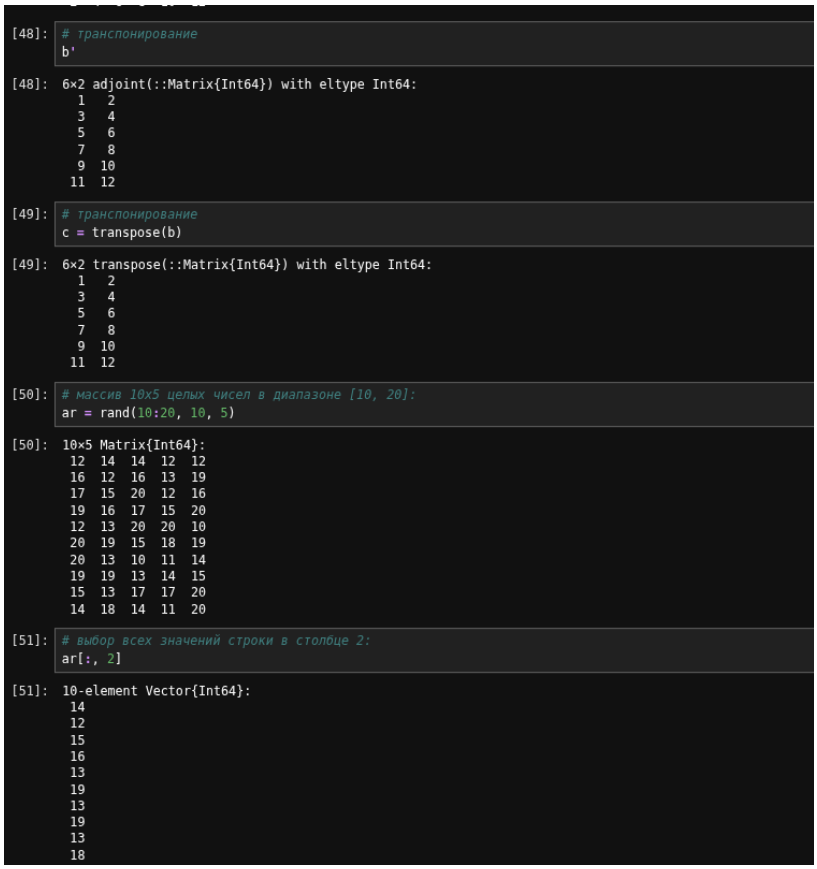

Рисунок 4.7: Примеры использования массивов

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

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

[58]: 10×5 BitMatrix:
       0  0  0  0  0
       1  0  1  0  1
       1  1  1  0  1
       1  1  1  1  1
       0  0  1  1  0
       1  1  1  1  1
       1  0  0  0  0
       1  1  0  0  1
       1  0  1  1  1
       0  1  0  0  1

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

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

Рисунок 4.8: Примеры использования массивов

Теперь перейдем к выполнению заданий.

Задание №1

Даны множества: $A = 0, 3, 4, 9, B = 1, 3, 4, 7, C = 0, 1, 2, 4, 7, 8, 9$. Найдем $P = A \cap B \cup A \cap B \cup A \cap C \cup B \cap C$ (рис. 4.9).

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

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

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

Рисунок 4.9: Задание №1. Работа с множествами

Задание №2

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

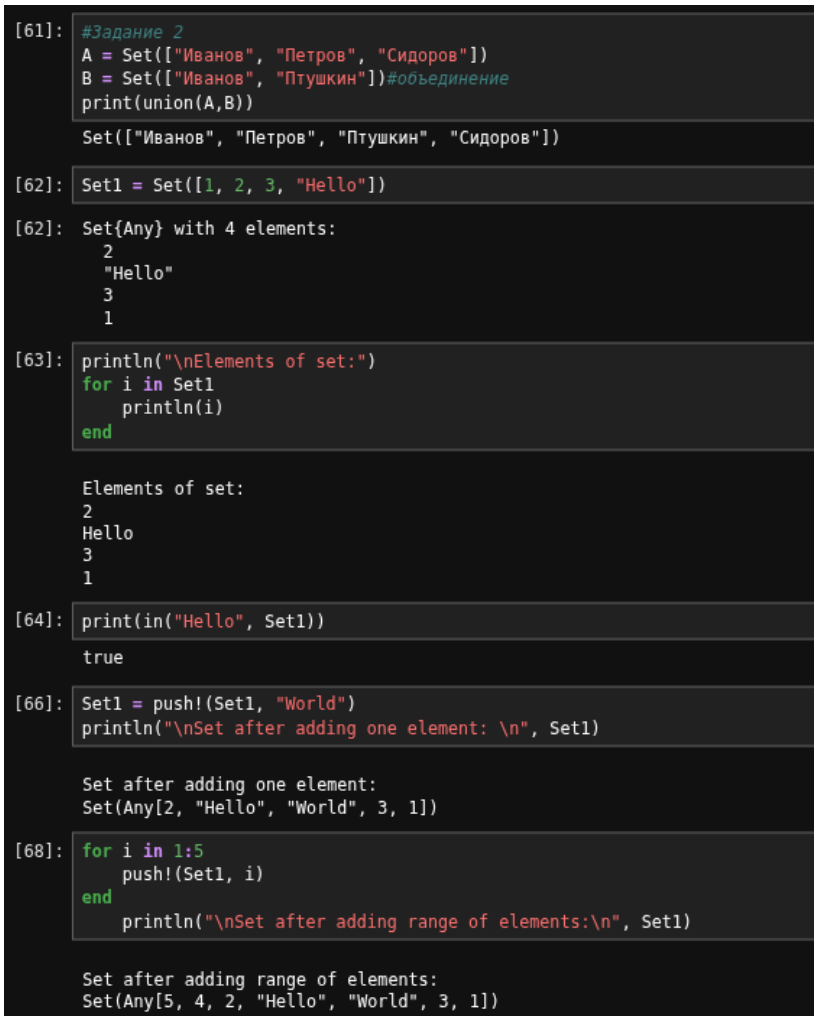

Рисунок 4.10: Задание №2. Примеры операций над множествами элементов разных типов

Задание №3

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

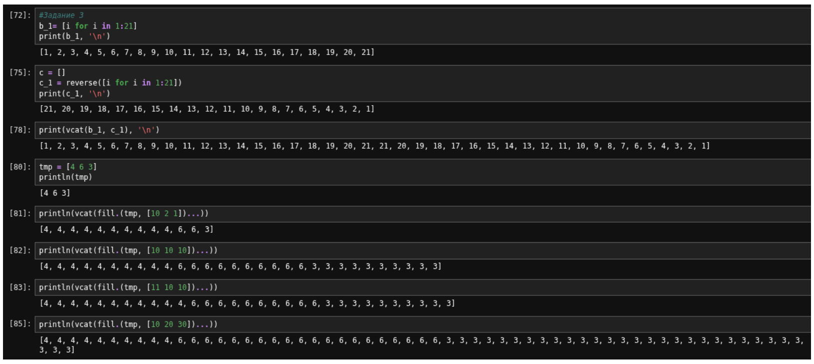

Рисунок 4.11: Задание №3. Работа с массивами

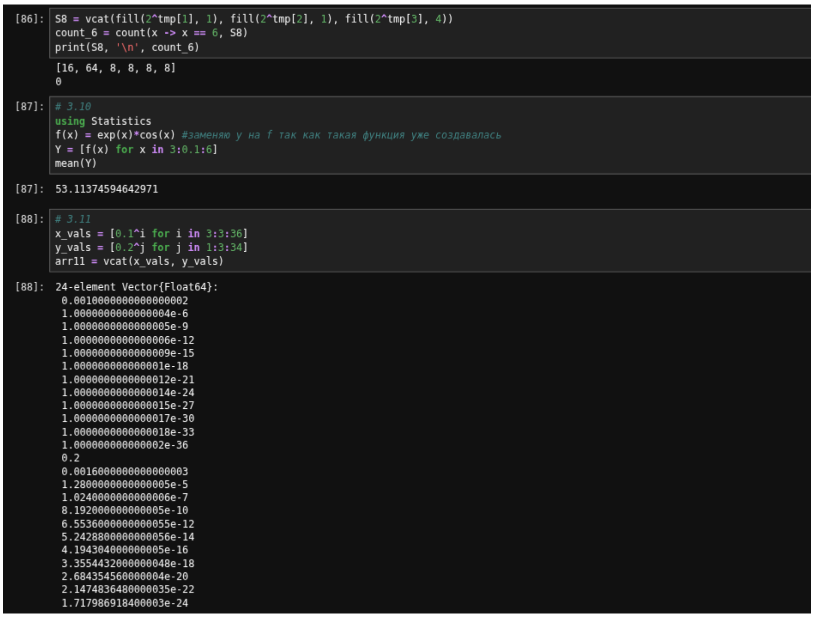

Рисунок 4.12: Задание №3. Работа с массивами

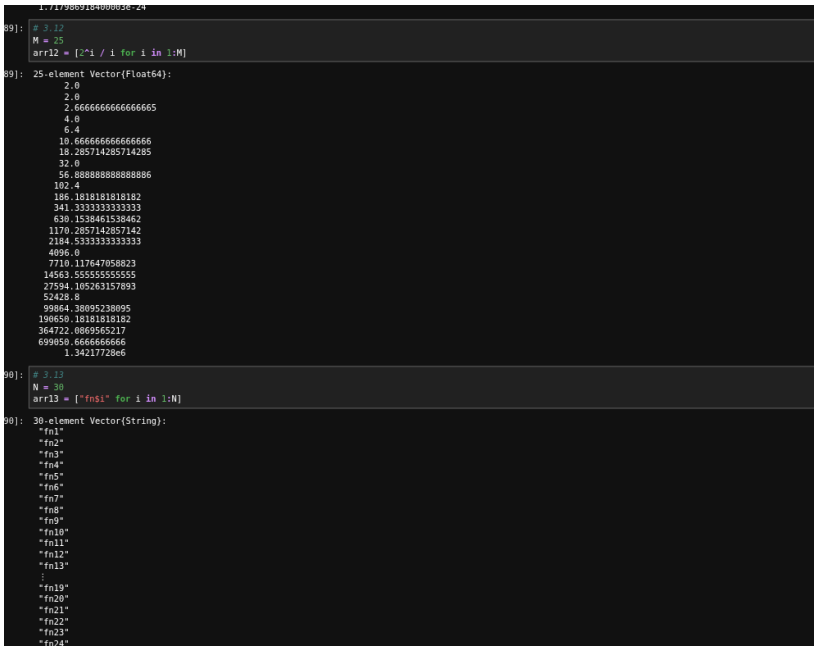

Рисунок 4.13: Задание №3. Работа с массивами

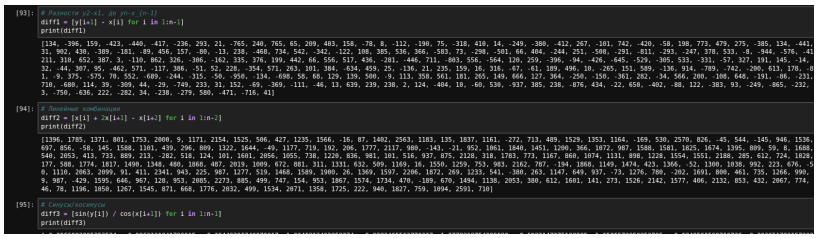

Рисунок 4.14: Задание №3. Работа с массивами

Рисунок 4.15: Задание №3. Работа с массивами

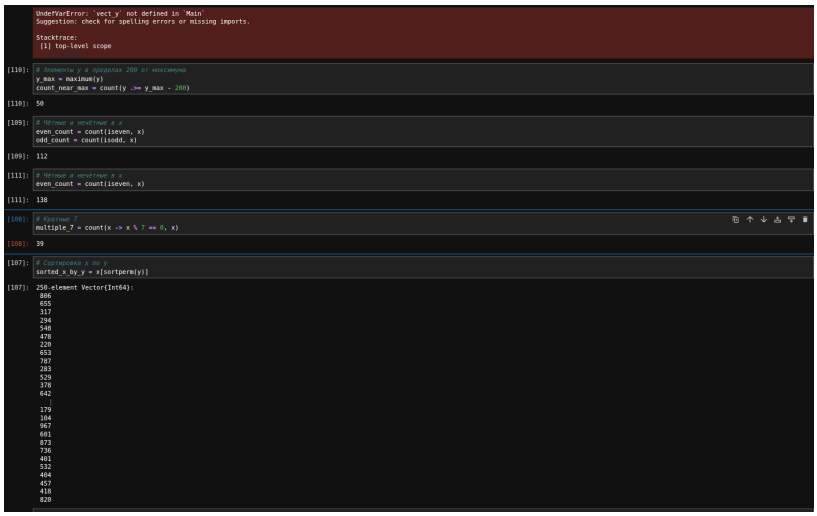

Рисунок 4.16: Задание №3. Работа с массивами

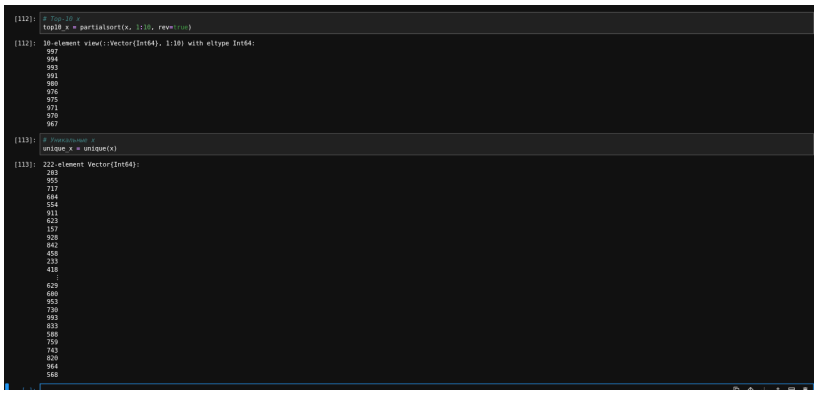

Рисунок 4.17: Задание №3. Работа с массивами

Задание №4

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

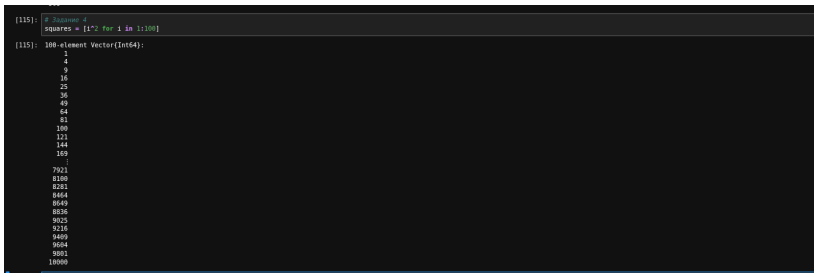

Рисунок 4.18: Задание №4

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

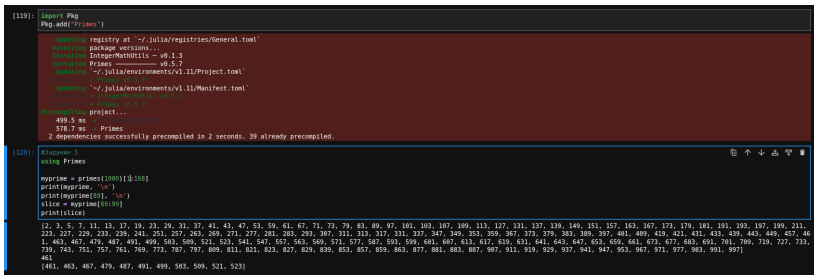

Рисунок 4.19: Задание №5. Работа с пакетом Primes

Задание №6

Вычислим следующие выражения (рис. 4.20).

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

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

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

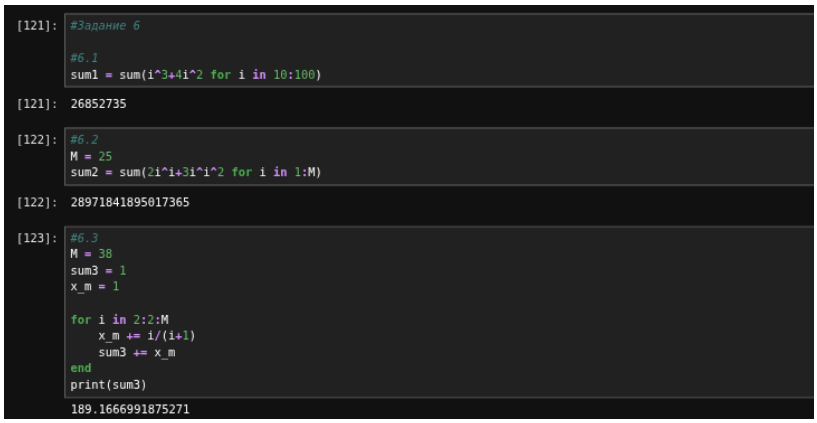

Рисунок 4.20: Задание №6

#Выводы

В результате выполнения данной лабораторной работы я изучил несколько структур дыннх, реализованных в Julia, Научился применять их и операции

над ними для решения задач.