

Лабораторная работа №3

Управляющие структуры

Дурдалыев Максат

Содержание

1	Введение	5
1.1	Цели и задачи	5
2	Выполнение лабораторной работы	6
2.1	Циклы while и for	6
2.2	Условные выражения	13
2.3	Функции	15
2.4	Сторонние библиотеки (пакеты) в Julia	24
2.5	Самостоятельная работа	27
3	Выводы	53
	Список литературы	54

Список иллюстраций

2.1	Примеры использования цикла while	7
2.2	Примеры использования цикла for	9
2.3	Пример использования цикла for для создания двумерного массива .	11
2.4	Пример использования цикла for для создания двумерного массива .	12
2.5	Примеры использования условного выражения	14
2.6	Примеры способов написания функции	16
2.7	Примеры способов написания функции	17
2.8	Сравнение результатов вывода	19
2.9	Примеры использования функций map() и broadcast()	21
2.10	Примеры использования функций map() и broadcast()	22
2.11	Примеры использования функций map() и broadcast()	23
2.12	Пример использования сторонних библиотек	26
2.13	Выполнение подпунктов задания №1	28
2.14	Выполнение подпунктов задания №1	29
2.15	Выполнение подпунктов задания №1	30
2.16	Выполнение задания №2 и №3	32
2.17	Выполнение задания №4	34
2.18	Выполнение задания №5	36
2.19	Выполнение задания №6	38
2.20	Выполнение задания №7	40
2.21	Выполнение задания №7	41
2.22	Выполнение задания №7	42
2.23	Выполнение задания №7 и №8	43
2.24	Выполнение подпунктов задания №8	45
2.25	Выполнение подпунктов задания №8	46
2.26	Выполнение задания №9 и №10	48
2.27	Выполнение подпунктов задания №10	50
2.28	Выполнение задания №11	52

Список таблиц

1 Введение

1.1 Цели и задачи

Цель работы

Основная цель работы — освоить применение циклов функций и сторонних для Julia пакетов для решения задач линейной алгебры и работы с матрицами[2].

Задание

1. Используя Jupyter Lab, повторите примеры.
2. Выполните задания для самостоятельной работы[1].

2 Выполнение лабораторной работы

2.1 Циклы while и for

Для различных операций, связанных с перебором индексируемых элементов структур данных, традиционно используются циклы while и for.

Синтаксис while

```
while <condition>  
    <loop body>  
end
```

Примеры использования цикла while (?@fig-001):

```
# пока n<10 прибавить к n единицу и распечатать значение:
n = 0
while n < 10
  n += 1
  println(n)
end
✓ 0.0s Julia
```

1
2
3
4
5
6
7
8
9
10

```
myfriends = ["Ted", "Robyn", "Barney", "Lily", "Marshall"]
i = 1
while i <= length(myfriends)
  friend = myfriends[i]
  println("Hi $friend, it's great to see you!")
  i += 1
end
✓ 0.0s Julia
```

Hi Ted, it's great to see you!
Hi Robyn, it's great to see you!
Hi Barney, it's great to see you!
Hi Lily, it's great to see you!
Hi Marshall, it's great to see you!

Рисунок 2.1: Примеры использования цикла while

Такие же результаты можно получить при использовании цикла for.

Синтаксис for

```
for <variable> in <range>  
    <loop body>  
end
```

Примеры использования цикла for (рис. [fig:002]):


```
for n in 1:2:10
println(n)
end
myfriends = ["Ted", "Robyn", "Barney", "Lily", "Marshall"]
for friend in myfriends
println("Hi $friend, it's great to see you!")
end
```

✓ 0.0s

Julia

1

3

5

7

9

Hi Ted, it's great to see you!

Hi Robyn, it's great to see you!

Hi Barney, it's great to see you!

Hi Lily, it's great to see you!

Hi Marshall, it's great to see you!

Рисунок 2.2: Примеры использования цикла for

Пример использования цикла `for` для создания двумерного массива, в котором значение каждой записи является суммой индексов строки и столбца (рис. [fig:003] - рис. [fig:004]):

```

# инициализация массива m x n из нулей:
m, n = 5, 5
A = fill{0, (m, n)}
# формирование массива, в котором значение каждой записи
# является суммой индексов строки и столбца:
for i in 1:m
    for j in 1:n
        A[i, j] = i + j
    end
end
A

```

✓ 0.0s

Julia

5x5 Matrix{Int64}:

```

2 3 4 5 6
3 4 5 6 7
4 5 6 7 8
5 6 7 8 9
6 7 8 9 10

```

```

# инициализация массива m x n из нулей:
B = fill{0, (m, n)}
for i in 1:m, j in 1:n
    B[i, j] = i + j
end
B

```

✓ 0.0s

Julia

5x5 Matrix{Int64}:

```

2 3 4 5 6
3 4 5 6 7
4 5 6 7 8
5 6 7 8 9
6 7 8 9 10

```

Рисунок 2.3: Пример использования цикла for для создания двумерного массива

```
C = [i + j for i in 1:m, j in 1:n]
```

```
C
```

```
✓ 0.0s
```

Julia

```
5x5 Matrix{Int64}:
```

```
2 3 4 5 6
```

```
3 4 5 6 7
```

```
4 5 6 7 8
```

```
5 6 7 8 9
```

```
6 7 8 9 10
```

Рисунок 2.4: Пример использования цикла for для создания двумерного массива

2.2 Условные выражения

Довольно часто при решении задач требуется проверить выполнение тех или иных условий. Для этого используют условные выражения.

Синтаксис условных выражений с ключевым словом:

```
if <condition 1>  
    <action 1>  
elseif <condition 2>  
    <action 2>  
else  
    <action 3>  
end
```

Примеры использования условного выражения (рис. [fig:005]):

```

# используем `&&` для реализации операции "AND"
# операция % вычисляет остаток от деления
N = 100
if (N % 3 == 0) && (N % 5 == 0)
println("FizzBuzz")
elseif N % 3 == 0
println("Fizz")
elseif N % 5 == 0
println("Buzz")
else
println(N)
end

```

✓ 0.0s

Julia

Buzz

```

x = 5
y = 10
(x > y) ? x : y

```

✓ 0.0s

Julia

10

Рисунок 2.5: Примеры использования условного выражения

2.3 Функции

Julia дает нам несколько разных способов написать функцию.

Примеры способов написания функции (рис. [fig:006] - рис. [fig:007]):

```
function sayhi(name)
println("Hi $name, it's great to see you!")
end
✓ 0.0s Julia

sayhi (generic function with 1 method)

# функция возведения в квадрат:
function f(x)
x^2
end
✓ 0.0s Julia

f (generic function with 1 method)

sayhi("C-3PO")
f(42)
✓ 0.0s Julia

Hi C-3PO, it's great to see you!

1764

sayhi2(name) = println("Hi $name, it's great to see you!")
f2(x) = x^2
✓ 0.0s Julia

f2 (generic function with 1 method)
```

Рисунок 2.6: Примеры способов написания функции


```
sayhi("C-3PO")
```

```
f(42)
```

✓ 0.0s

Julia

Hi C-3PO, it's great to see you!

1764

```
sayhi3 = name -> println("Hi $name, it's great to see you!")
```

```
f3 = x -> x^2
```

✓ 0.0s

Julia

#83 (generic function with 1 method)

```
sayhi("C-3PO")
```

```
f(42)
```

✓ 0.0s

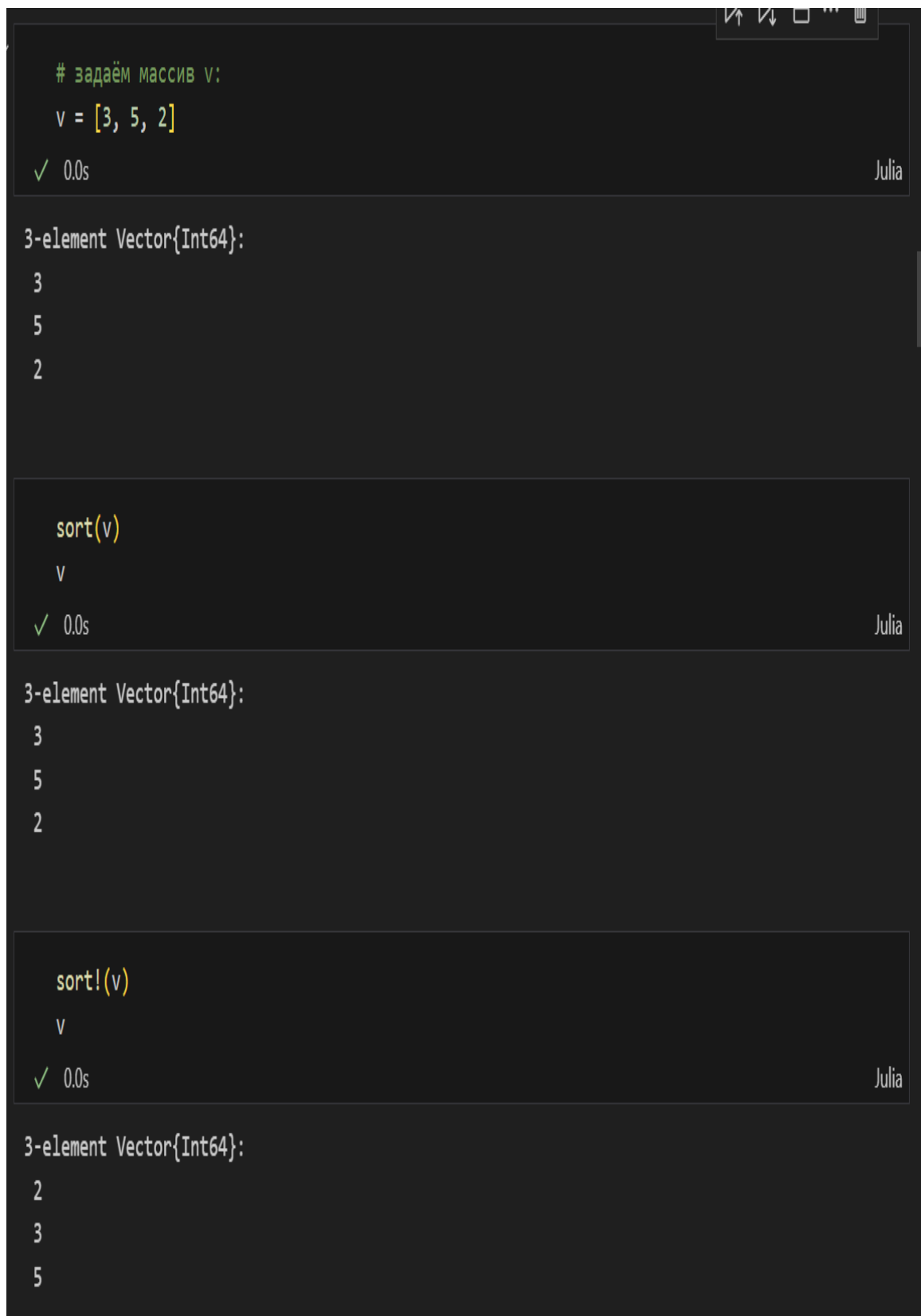
Julia

Hi C-3PO, it's great to see you!

1764

Рисунок 2.7: Примеры способов написания функции

По соглашению в Julia функции, сопровождаемые восклицательным знаком, изменяют свое содержимое, а функции без восклицательного знака не делают этого (рис. [fig:008]):



```
# задаём массив v:  
v = [3, 5, 2]  
✓ 0.0s Julia  
  
3-element Vector{Int64}:  
 3  
 5  
 2  
  
sort(v)  
v  
✓ 0.0s Julia  
  
3-element Vector{Int64}:  
 3  
 5  
 2  
  
sort!(v)  
v  
✓ 0.0s Julia  
  
3-element Vector{Int64}:  
 2  
 3  
 5
```

Рисунок 2.8: Сравнение результатов вывода

В Julia функция `map` является функцией высшего порядка, которая принимает функцию в качестве одного из своих входных аргументов и применяет эту функцию к каждому элементу структуры данных, которая ей передаётся также в качестве аргумента.

Функция `broadcast` — ещё одна функция высшего порядка в Julia, представляющая собой обобщение функции `map`. Функция `broadcast()` будет пытаться привести все объекты к общему измерению, `map()` будет напрямую применять данную функцию поэлементно.

Примеры использования функций `map()` и `broadcast()` (рис. [fig:009] - рис. [fig:011]):



Рисунок 2.9: Примеры использования функций `map()` и `broadcast()`

```
f.([1, 2, 3])
✓ 0.0s Julia

3-element Vector{Int64}:
 1
 4
 9
```

▶ ▶ ▢ ... 🗑

```
# Задаём матрицу A:
A = [i + 3*j for j in 0:2, i in 1:3]
✓ 0.0s Julia

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

```
f(A)
✓ 0.0s Julia

3×3 Matrix{Int64}:
30  36  42
66  81  96
102 126 150
```

```
B = f.(A)
✓ 0.0s Julia

3×3 Matrix{Int64}:
 1  4  9
16 25 36
49 64 81
```

Рисунок 2.10: Примеры использования функций `map()` и `broadcast()`

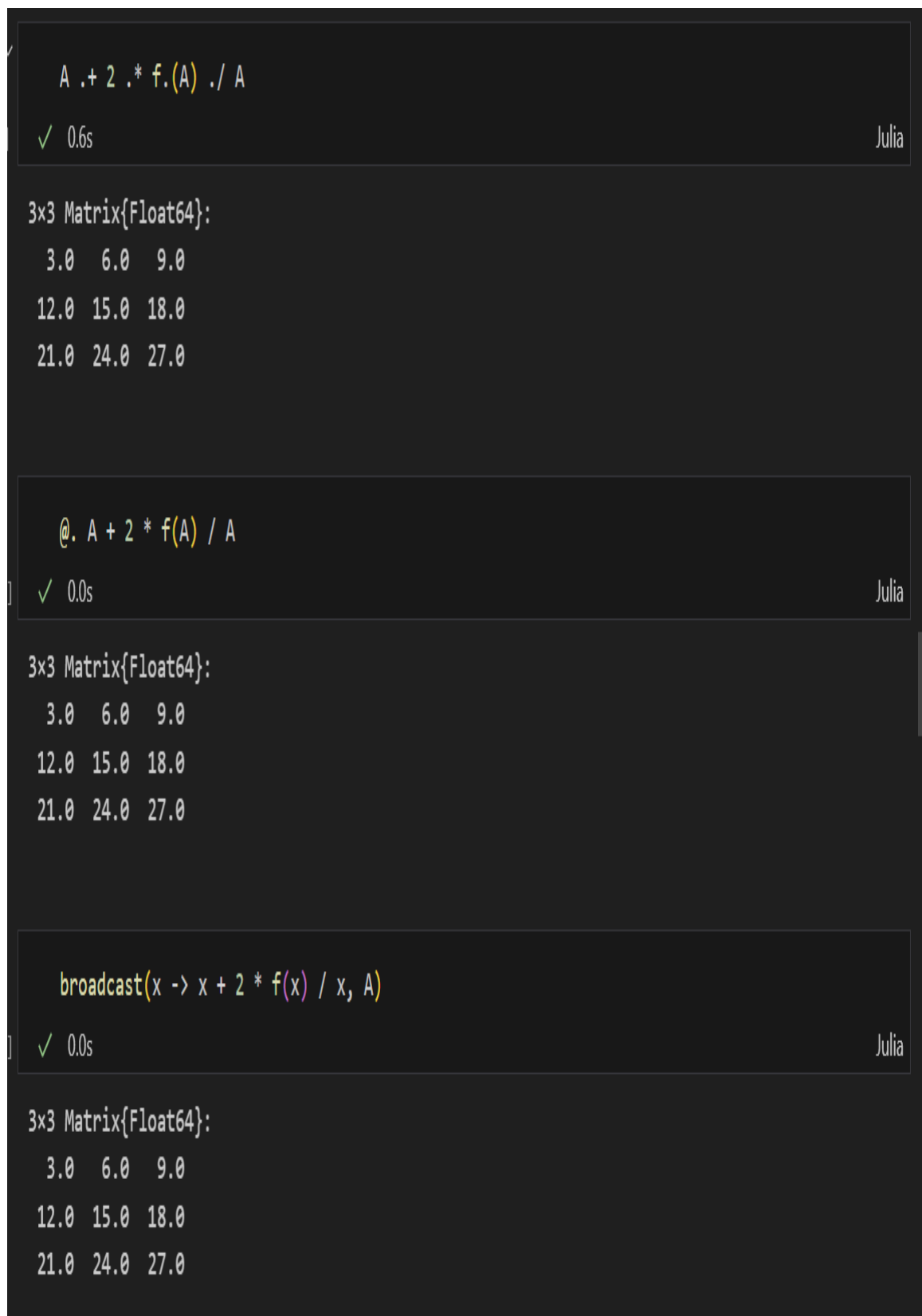


Рисунок 2.11: Примеры использования функций `map()` и `broadcast()`

2.4 Сторонние библиотеки (пакеты) в Julia

Julia имеет более 2000 зарегистрированных пакетов, что делает их огромной частью экосистемы Julia. Есть вызовы функций первого класса для других языков, обеспечивающие интерфейсы сторонних функций. Можно вызвать функции из Python или R, например, с помощью PyCall или Rcall.

С перечнем доступных в Julia пакетов можно ознакомиться на страницах следующих ресурсов: - <https://julialang.org/packages/> - <https://juliahub.com/ui/Home> - <https://juliaobserver.com/> - <https://github.com/svaksha/Julia.jl>

При первом использовании пакета в вашей текущей установке Julia вам необходимо использовать менеджер пакетов, чтобы явно его добавить:

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

При каждом новом использовании Julia (например, в начале нового сеанса в REPL или открытии блокнота в первый раз) нужно загрузить пакет, используя ключевое слово `using`:

Например, добавим и загрузим пакет `Colors`:

```
Pkg.add("Colors")
using Colors
```

Затем создадим палитру из 100 разных цветов:

```
palette = distinguishable_colors(100)
```

А затем определим матрицу 3×3 с элементами в форме случайного цвета из палитры, используя функцию `rand`:


```
rand(palette, 3, 3)
```

Пример использования сторонних библиотек (рис. [fig:012]):

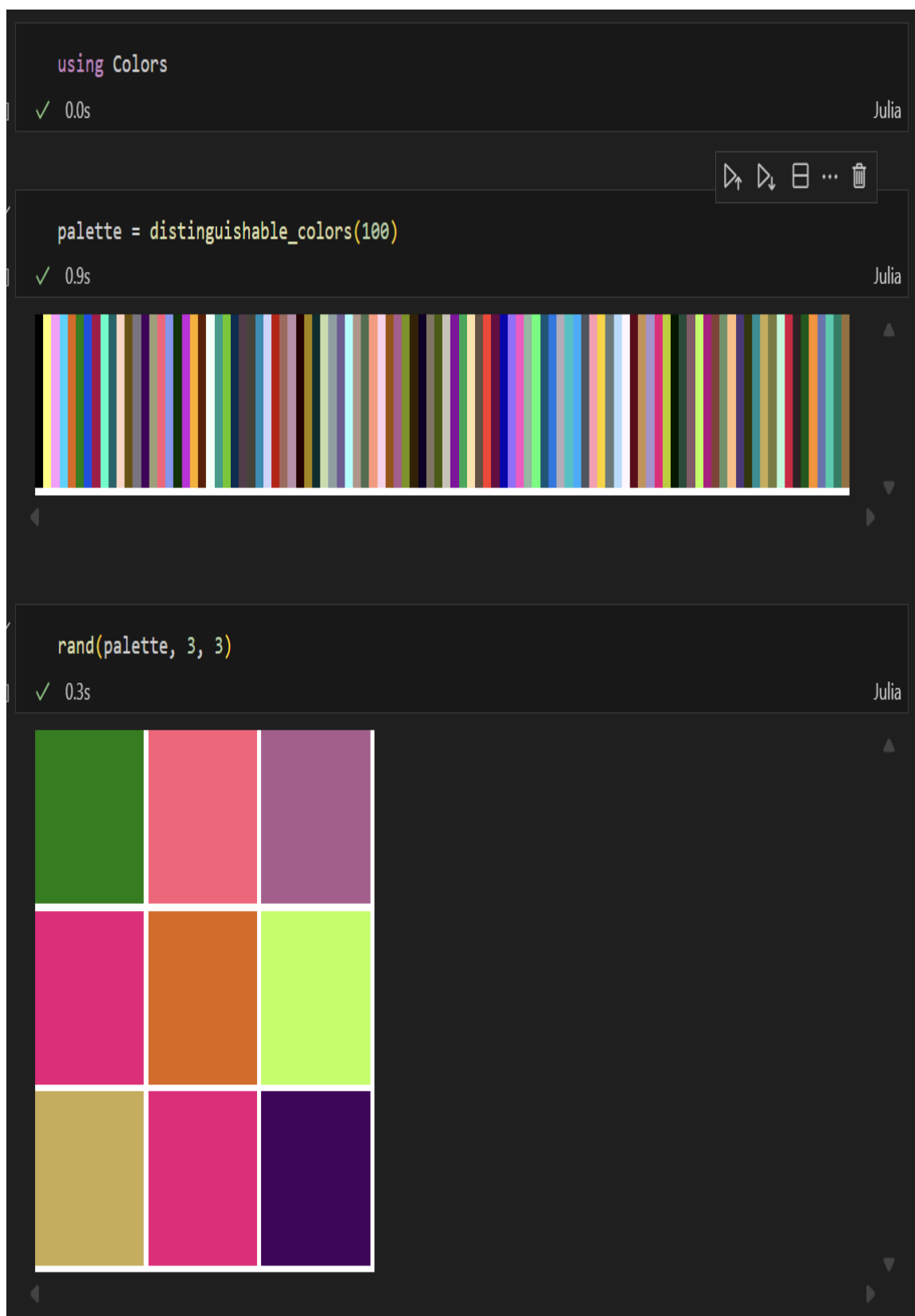


Рисунок 2.12: Пример использования сторонних библиотек

2.5 Самостоятельная работа

Выполнение задания №1 (рис. [fig:013] - рис. [fig:015]):

Задания для самостоятельного выполнения

№1. Используя циклы `while` и `for`:

1.1) выведите на экран целые числа от 1 до 100 и напечатайте их квадраты:

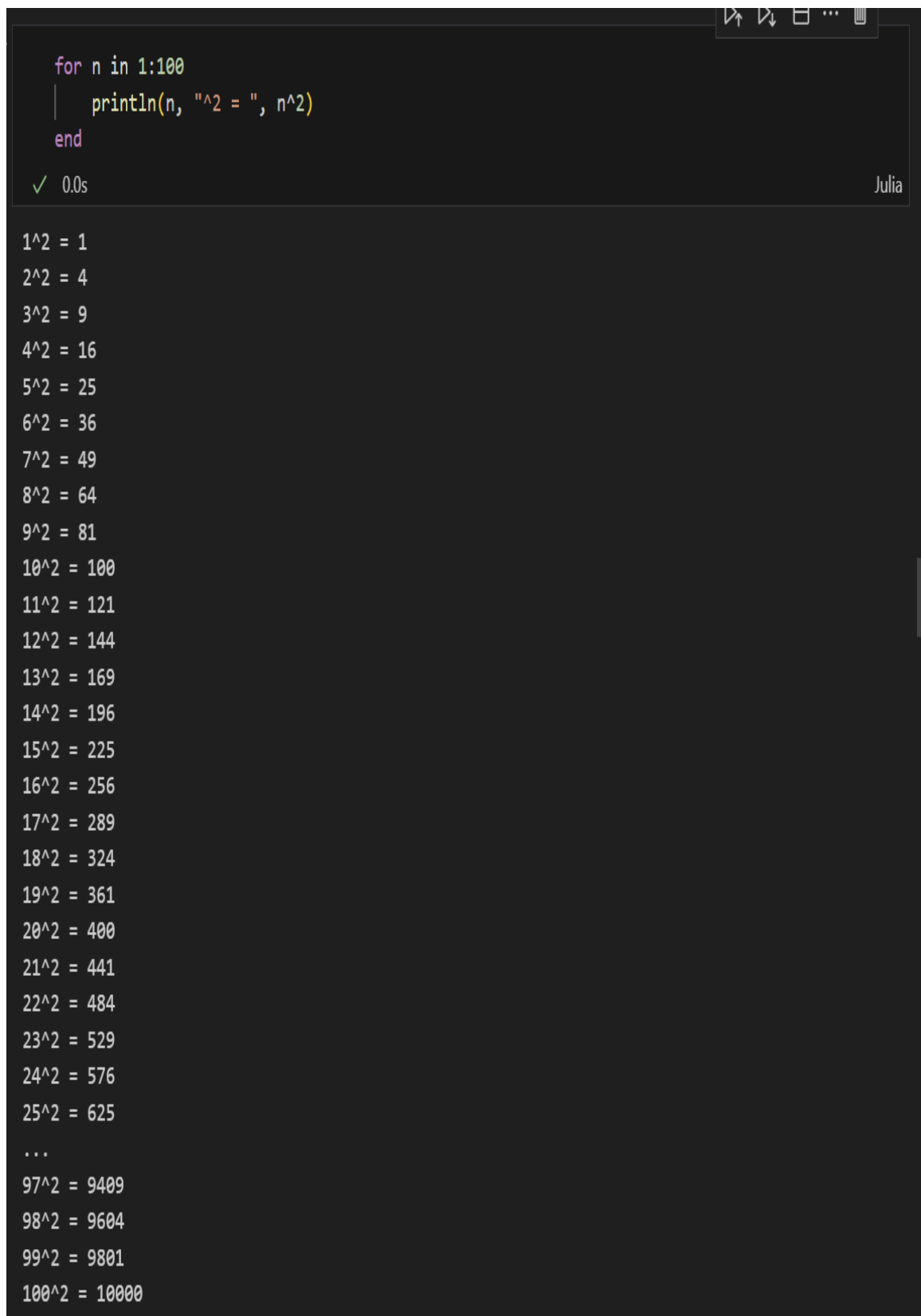
```
n = 1
while n <= 100
    println(n, "^2 = ", (n^2))
    n += 1
end
```

✓ 0.0s

Julia

```
1^2 = 1
2^2 = 4
3^2 = 9
4^2 = 16
5^2 = 25
6^2 = 36
7^2 = 49
8^2 = 64
9^2 = 81
10^2 = 100
11^2 = 121
12^2 = 144
13^2 = 169
14^2 = 196
15^2 = 225
16^2 = 256
17^2 = 289
18^2 = 324
19^2 = 361
20^2 = 400
21^2 = 441
22^2 = 484
23^2 = 529
24^2 = 576
25^2 = 625
...
97^2 = 9409
98^2 = 9604
99^2 = 9801
100^2 = 10000
```

Рисунок 2.13: Выполнение подпунктов задания №1



```
for n in 1:100
    println(n, "^2 = ", n^2)
end
```

✓ 0.0s Julia

1^2 = 1
2^2 = 4
3^2 = 9
4^2 = 16
5^2 = 25
6^2 = 36
7^2 = 49
8^2 = 64
9^2 = 81
10^2 = 100
11^2 = 121
12^2 = 144
13^2 = 169
14^2 = 196
15^2 = 225
16^2 = 256
17^2 = 289
18^2 = 324
19^2 = 361
20^2 = 400
21^2 = 441
22^2 = 484
23^2 = 529
24^2 = 576
25^2 = 625
...
97^2 = 9409
98^2 = 9604
99^2 = 9801
100^2 = 10000

Рисунок 2.14: Выполнение подпунктов задания №1

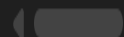
1.2) создайте словарь `squares`, который будет содержать целые числа в качестве ключей и квадраты в качестве их пар-значений:

```
squares = Dict()
for n in 1:100
    squares[n] = n^2
end
println(squares)
```

✓ 0.4s

Julia

Dict{Any, Any}{5 => 25, 56 => 3136, 35 => 1225, 55 => 3025, 60 => 3600, 30 => 900, 32 => 1024, 6



1.3) создайте массив `squares_arr`, содержащий квадраты всех чисел от 1 до 100:

```
squares_arr = [n^2 for n in 1:100]
println(squares_arr)
```

✓ 0.1s

Julia

[1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361, 400, 441, 48

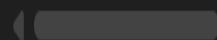


Рисунок 2.15: Выполнение подпунктов задания №1

Выполнение задания №2 и №3 (рис. [fig:016]):

№2. Напишите условный оператор, который печатает число, если число чётное, и строку «нечётное», если число нечётное.

Перепишите код, используя тернарный оператор:

```
n = 17
if n % 2 == 0
|   println(n)
else
|   println("Нечётное")
end
```

✓ 0.0s

Julia

Нечётное

```
n = 12
println(n % 2 == 0 ? n : "Нечётное")
```

✓ 0.0s

Julia

12

№3. Напишите функцию add_one, которая добавляет 1 к своему входу:

```
function add_one(x)
|   return x + 1
end
println(add_one(12))
```

✓ 0.0s

Julia

13

Рисунок 2.16: Выполнение задания №2 и №3

Выполнение задания №4 (рис. [fig:017]):

№4. Используйте `map()` или `broadcast()` для задания матрицы A, каждый элемент которой увеличивается на единицу по сравнению с предыдущим:

```
A = [12 11 10; 12 11 10; -12 -11 -10]
B = map(x -> x + 1, A)
println(B)
```

✓ 0.0s

Julia

```
[13 12 11; 13 12 11; -11 -10 -9]
```

```
B = broadcast(x -> x + 1, A)
println(B)
```

✓ 0.0s

Julia

```
[13 12 11; 13 12 11; -11 -10 -9]
```

Рисунок 2.17: Выполнение задания №4

Выполнение задания №5 (рис. [fig:018]):

№5. Задайте матрицу A следующего вида. Найдите A^3 . Замените третий столбец матрицы A на сумму второго и третьего столбцов:

```
A = [1 1 3; 5 2 6; -2 -1 -3]
println(map(x -> x^3, A))
```

✓ 0.0s

Julia

```
[1 1 27; 125 8 216; -8 -1 -27]
```

```
A
```

✓ 0.0s

Julia

```
3x3 Matrix{Int64}:
```

```
1  1  3
5  2  6
-2 -1 -3
```

```
A[:, 3] = A[:, 2] + A[:, 3]
```

```
A
```

✓ 0.0s

Julia

```
3x3 Matrix{Int64}:
```

```
1  1  4
5  2  8
-2 -1 -4
```

Рисунок 2.18: Выполнение задания №5

Выполнение задания №6 (рис. [fig:019]):

№6. Создайте матрицу B с элементами $B_{i1} = 10$, $B_{i2} = -10$, $B_{i3} = 10$, $i = 1, 2, \dots, 15$. Вычислите матрицу $C = B^T B$:

```
B = repeat([10 -10 10], 15, 1)
```

✓ 0.1s

Julia

15×3 Matrix{Int64}:

```
10 -10 10
10 -10 10
10 -10 10
10 -10 10
10 -10 10
10 -10 10
10 -10 10
10 -10 10
10 -10 10
10 -10 10
10 -10 10
10 -10 10
10 -10 10
10 -10 10
10 -10 10
```

```
C = B' * B
println(C)
```

✓ 0.0s

Julia

```
[1500 -1500 1500; -1500 1500 -1500; 1500 -1500 1500]
```

Рисунок 2.19: Выполнение задания №6

Выполнение задания №7 (рис. [fig:020] - рис. [fig:023]):

№7. Создайте матрицу Z размерности 6 x 6, все элементы которой равны нулю и матрицу E, все элементы которой равны 1. Используя цикл while или for и закономерности расположения элементов, создайте следующие матрицы размерности 6 x 6:

```
Z = zeros(Int, 6, 6)
println(Z)
E = ones(Int, 6, 6)
println(E)
n = 6

Z1 = copy(Z)
for i in 1:n
    for j in 1:n
        if abs(i - j) == 1
            Z1[i, j] = 1
        end
    end
end
println(Z1)

Z2 = copy(Z)
for i in 1:n
    if i + 2 < n
        Z2[i, i] = 1
        Z2[i, i+2] = 1
    elseif (i == 3) || (i == 4)
        Z2[i, i] = 1
        Z2[i, i+2] = 1
        Z2[i, i-2] = 1
    else
        Z2[i, i] = 1
        Z2[i, i-2] = 1
    end
end
println(Z2)
```

Рисунок 2.20: Выполнение задания №7


```

Z3 = copy(Z)
for i in 1:n
    for j in 1:n
        if (i + j) % 2 != 0
            Z3[i, j] = 1
        end
    end
end
println(Z3)

```

```

Z4 = copy(Z)
for i in 1:n
    for j in 1:n
        if (i + j) % 2 == 0
            Z4[i, j] = 1
        end
    end
end
println(Z4)

```

n::Int64 = 6

✓ 0.0s

Julia

```

[0 0 0 0 0; 0 0 0 0 0; 0 0 0 0 0; 0 0 0 0 0; 0 0 0 0 0; 0 0 0 0 0]
[1 1 1 1 1; 1 1 1 1 1; 1 1 1 1 1; 1 1 1 1 1; 1 1 1 1 1; 1 1 1 1 1]
[0 1 0 0 0; 1 0 1 0 0; 0 1 0 1 0; 0 0 1 0 1; 0 0 0 1 0; 0 0 0 1 0]
[1 0 1 0 0; 0 1 0 1 0; 0 0 1 0 1; 0 1 0 1 1; 0 0 1 0 1; 0 0 0 1 0]
[0 1 0 1 1; 1 0 1 0 1; 0 1 0 1 1; 1 0 1 0 1; 0 1 0 1 1; 1 0 1 0 1]
[1 0 1 0 1; 0 1 0 1 1; 1 0 1 0 1; 0 1 0 1 1; 1 0 1 0 1; 0 1 0 1 1]

```

Рисунок 2.21: Выполнение задания №7

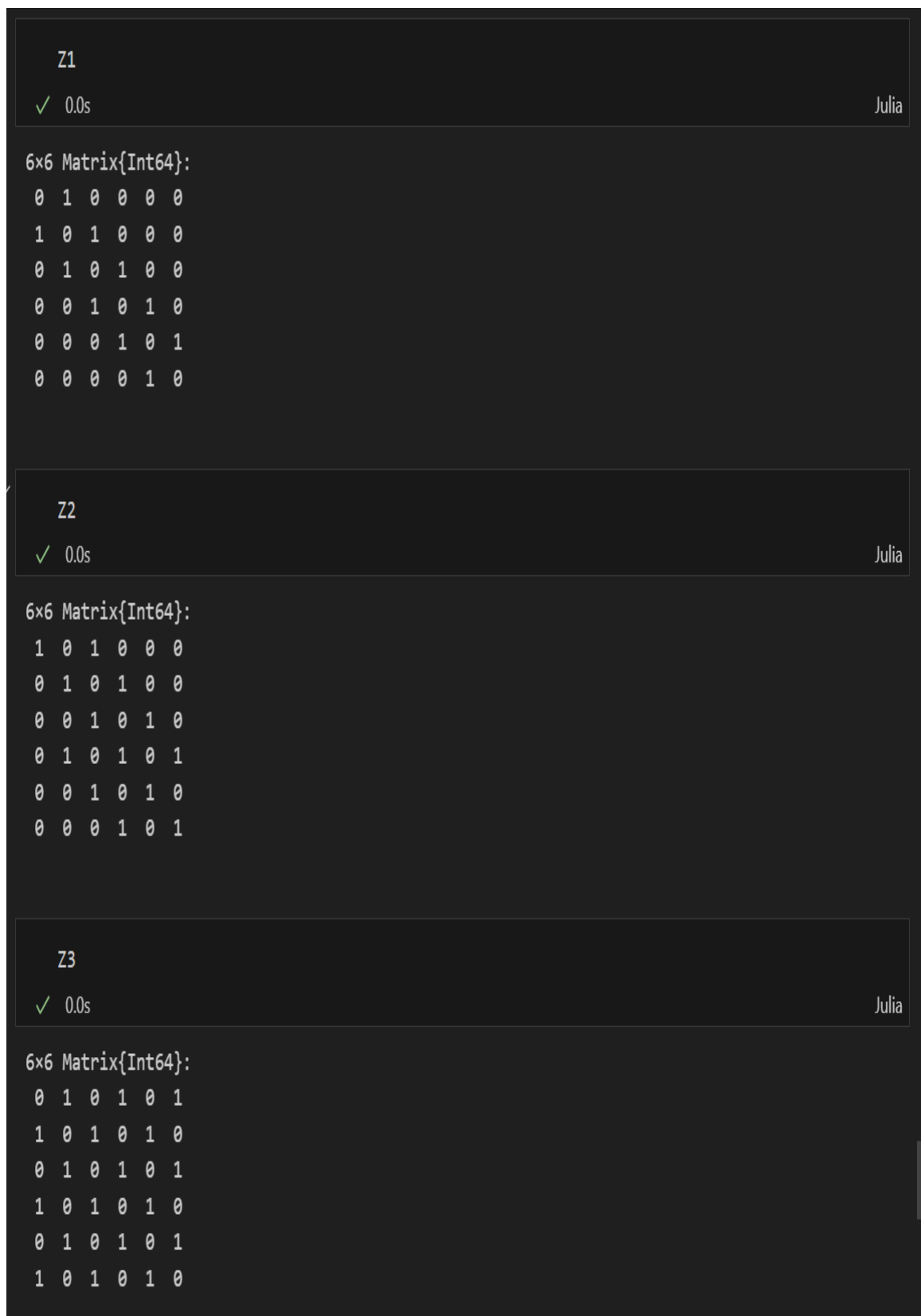


Рисунок 2.22: Выполнение задания №7

Z4

✓ 0.0s

Julia

```

6×6 Matrix{Int64}:
 1  0  1  0  1  0
 0  1  0  1  0  1
 1  0  1  0  1  0
 0  1  0  1  0  1
 1  0  1  0  1  0
 0  1  0  1  0  1

```

Создать

+ Code

+ Markdown

№8. В языке R есть функция `outer()`. Фактически, это матричное умножение с возможностью изменить применяемую операцию (например, заменить произведение на сложение или возведение в степень):

8.1) Напишите свою функцию, аналогичную функции `outer()` языка R. Функция должна иметь следующий интерфейс: `outer(x,y,operation)`:

```

function outer(x, y, operation)
|   return [operation(xi, yj) for xi in x, yj in y]
end

```

✓ 0.0s

Julia

outer (generic function with 1 method)

Рисунок 2.23: Выполнение задания №7 и №8

Выполнение задания №8 (рис. [fig:024] - рис. [fig:025]):

8.2) Используя написанную вами функцию `outer()`, создайте матрицы следующей структуры:

```
A1 = outer(0:4, 0:4, +)
```

A1

✓ 0.0s

Julia

5×5 Matrix{Int64}:

```
0 1 2 3 4
1 2 3 4 5
2 3 4 5 6
3 4 5 6 7
4 5 6 7 8
```

```
A2 = outer(0:4, 1:5, ^)
```

A2

✓ 0.0s

Julia

5×5 Matrix{Int64}:

```
0 0 0 0 0
1 1 1 1 1
2 4 8 16 32
3 9 27 81 243
4 16 64 256 1024
```

```
A3 = outer(0:4, 0:4, (x, y) -> mod(x+y, 5))
```

A3

✓ 0.0s

Julia

5×5 Matrix{Int64}:

```
0 1 2 3 4
1 2 3 4 0
2 3 4 0 1
3 4 0 1 2
4 0 1 2 3
```

Рисунок 2.24: Выполнение подпунктов задания №8

```
A4 = outer(0:9, 0:9, (x, y) -> mod(x+y,10))
```

```
A4
```

✓ 0.0s
Julia

```
10x10 Matrix{Int64}:
 0  1  2  3  4  5  6  7  8  9
 1  2  3  4  5  6  7  8  9  0
 2  3  4  5  6  7  8  9  0  1
 3  4  5  6  7  8  9  0  1  2
 4  5  6  7  8  9  0  1  2  3
 5  6  7  8  9  0  1  2  3  4
 6  7  8  9  0  1  2  3  4  5
 7  8  9  0  1  2  3  4  5  6
 8  9  0  1  2  3  4  5  6  7
 9  0  1  2  3  4  5  6  7  8
```

✦ Создать
+ Code
+ Markdown

```
A5 = outer(0:8, 0:8, (x, y) -> mod(x-y,9))
```

```
A5
```

✓ 0.0s
Julia

```
9x9 Matrix{Int64}:
 0  8  7  6  5  4  3  2  1
 1  0  8  7  6  5  4  3  2
 2  1  0  8  7  6  5  4  3
 3  2  1  0  8  7  6  5  4
 4  3  2  1  0  8  7  6  5
 5  4  3  2  1  0  8  7  6
 6  5  4  3  2  1  0  8  7
 7  6  5  4  3  2  1  0  8
 8  7  6  5  4  3  2  1  0
```

Рисунок 2.25: Выполнение подпунктов задания №8

Выполнение задания №9 и №10(рис. [fig:026]):

№9. Решите следующую систему линейных уравнений с 5 неизвестными:

```
A = [1 2 3 4 5;  
     2 1 2 3 4;  
     3 2 1 2 3;  
     4 3 2 1 2;  
     5 4 3 2 1]  
b = [7; -1; -3; 5; 17]
```

```
x = A \ b  
println(x)
```

✓ 0.1s

Julia

[-2.0000000000000036, 3.0000000000000058, 4.999999999999998, 1.999999999999991, -3.999999999999999]

№10. Создайте матрицу M размерности 6 x 10, элементами которой являются целые числа, выбранные случайным образом с повторениями из совокупности 1, 2, ..., 10:

```
M = rand(1:10, 6, 10)  
M
```

✓ 0.0s

Julia

6x10 Matrix{Int64}:

4	1	3	7	6	9	3	7	5	4
5	9	1	8	5	7	9	7	2	6
7	6	4	3	1	3	9	4	10	10
1	1	7	6	2	5	7	7	7	7
8	1	6	3	9	1	10	6	1	5
6	7	3	5	6	7	9	7	10	7

Рисунок 2.26: Выполнение задания №9 и №10

Выполнение задания №10 (рис. [fig:027]):

10.1) Найдите число элементов в каждой строке матрицы M, которые больше числа N (например, N = 4):

```
N = 4
greater_than_N = sum(M .> N)
println(greater_than_N)
```

✓ 0.0s

Julia

40

10.2) Определите, в каких строках матрицы M число M (например, M = 7) встречается ровно 2 раза:

```
rows_with_M_twice = [i for i=1:6 if sum(M[i, :] .== 7) == 2]
println(rows_with_M_twice)
```

✓ 0.0s

Julia

[1, 2]

10.3) Определите все пары столбцов матрицы M, сумма элементов которых больше K (например, K = 75):

```
K = 75
col_pairs = []
for i in 1:size(M, 2)-1
    for j in i+1:size(M, 2)
        if sum(M[:,i] .+ M[:,j]) > K
            push!(col_pairs, (i, j))
        end
    end
end
println(col_pairs)
```

✓ 0.2s

Julia

Any[(1, 7), (4, 7), (5, 7), (6, 7), (7, 8), (7, 9), (7, 10), (8, 10)]

Рисунок 2.27: Выполнение подпунктов задания №10

Выполнение задания №11 (рис. [fig:028]):

№11. Вычислите:

```
sum_1 = sum(i^4 / (3 + j) for i in 1:20 for j in 1:5)
println(sum_1)
```

✓ 0.0s

Julia

639215.2833333334

```
sum_2 = sum(i^4 / (3 + i * j) for i in 1:20 for j in 1:5)
println(sum_2)
```

✓ 0.0s

Julia

89912.02146097136

Рисунок 2.28: Выполнение задания №11

3 Выводы

В результате выполнения данной лабораторной работы я освоил применение циклов функций и сторонних для Julia пакетов для решения задач линейной алгебры и работы с матрицами.

Список литературы

- [1] *Julia 1.11 Documentation*. URL: <https://docs.julialang.org/en/v1/>.
- [2] *JuliaLang*. URL: <https://julialang.org/>.