Лабораторная работа № 2

Дисциплина: Компьютерный практикум по статистическому анализу данных

Покрас Илья Михайлович

Содержание

Список иллюстраций

1 Цель работы

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

2 Выполнение работы

1. С помощью intersect и union я посчитал значение P, равное объединению 2 пересеченИЙ A, B A, C и B, C (2.1).

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

P = union(intersect(A,B), intersect(A,B), intersect(A,C), intersect(B,C))
println("P equals: ")
println(P)
```

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

Рис. 2.1: Пункт 1

2. С помощью setdiff я вывел элементы множества, входящие в Set1 и не входящие в Set2. С помощью issetequal я проверил, являются ли set_bool1 и set_bool2 одинаковыми (2.2).

```
Set1 = Set(["Hello", "World", 1, 2, 3])
Set2 = Set([1, 2, 3, 4])
Set_bool1 = Set([true, false, false, true])
Set_bool2 = Set([true, false, false, true])

println("Set1:")
println(Set1)
println("\nSet2:")
println(Set2)

println("\n'setdiff' Operation:")
println(setdiff(Set1, Set2))
println("\n'intersect' Operation:")
println(intersect(Set1, Set2))

println("\n'issetequal' Operation:")
println(issetequal(Set_bool1, Set_bool2))
```

```
Set1:
Set(Any["Hello", 2, "World", 3, 1])

Set2:
Set([4, 2, 3, 1])

'setdiff' Operation:
Set(Any["Hello", "World"])

'intersect' Operation:
Set(Any[2, 3, 1])

'issetequal' Operation:
true
```

Рис. 2.2: Пункт 2

3. С помощью vcat и collect я создал массив с длиной N, значения которого равны $1, 2, 3, \ldots, N$ (2.3).

```
N = 30

arr1 = collect(1:N)
arr2 = vcat(1:N-1, N)

println("Vector1:")
println(arr1)

println("\nVector1:")
println(arr2)
```

```
Vector1:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30]

Vector1:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30]
```

Рис. 2.3: Пункт 3.1

4. С помощью collect и шага -1, reverse я создал массив длиной N вида $N, N-1, N-2, \ldots, 1$ (2.4).

```
N = 25

arr_reverse1 = collect(N:-1:1)
arr_reverse2 = reverse(collect(1:N))


println("Reversed Vector1:")
println(arr_reverse1)

println("\nReversed Vector2:")
println(arr_reverse2)
```

```
Reversed Vector1:
[25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

Reversed Vector2:
[25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

Рис. 2.4: Пункт 3.2

5. Используя collect, я создал 2 половины массива, после чего с помощью vcat я объединил их и N, чтобы получить массив вида $1, 2, 3, \ldots, N, N-1, N-2, \ldots, 1$ (2.5).

```
N = 20

arr_half1 = collect(1:N-1)
arr_half2 = collect(N-1:-1:1)
arr_combined1 = vcat(arr_half1, N, arr_half2)
arr_combined2 = [1:N; N-1:-1:1]

println("Combined Vector1:")
println(arr_combined1)

println("\nCombined Vector2:")
println(arr_combined2)
```

```
Combined Vector1:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

Combined Vector2:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

Рис. 2.5: Пункт 3.3

6. Я создал массив tmp вида [4, 6, 3] (2.6).

```
tmp1 = [4, 6, 3]
tmp2 = [x for x in [4, 6, 3]]

println("tmp1:")
println(tmp1)

println("\ntmp2:")
println(tmp2)
```

```
tmp1:
[4, 6, 3]

tmp2:
[4, 6, 3]
```

Рис. 2.6: Пункт 3.4

7. Я создал массив, в котором первый элемент массива tmp повторяется 10 раз с помощью циклов for, push! и vcat(2.7).

```
tmp_filled1 = []
tmp_filled2 = tmp1[1]

foreach(_ -> push!(tmp_filled1, tmp1[1]), 1:10)
for i in 1:10 tmp_filled2 = vcat(tmp2[1], tmp_filled2) end


println("Filled tmp1:")
println(tmp_filled1)

println("\nFilled tmp2:")
println(tmp_filled2)
```

```
Filled tmp1:
Any[4, 4, 4, 4, 4, 4, 4, 4, 4, 4]

Filled tmp2:
[4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4]
```

Рис. 2.7: Пункт 3.5

8. Я создал массив, в котором все элементы массива tmp повторяются 10 раз, использся foreach с push! и fill (2.8).

```
rep_tmp1 = tmp1
rep_tmp2 = repeat(tmp2, inner=10)

for i in 1:9 rep_tmp1 = vcat(tmp1, rep_tmp1) end
println("Repeated tmp1:")
println(rep_tmp1)

println("\nRepeated tmp2:")
println(rep_tmp2)
```

```
Repeated tmp1:
[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]

Repeated tmp2:
[4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3]
```

Рис. 2.8: Пункт 3.6

9. Я создал массив, в котором первый элемент массива tmp встречается 11 раз, второй элемент — 10 раз, третий элемент — 10 раз (2.9).

```
rep_tmp3 = []
rep_tmp4 = [fill(tmp1[1], 11); fill(tmp1[2], 10); fill(tmp1[3], 10)]

foreach(_ -> push!(rep_tmp3, tmp2[1]), 1:11); foreach(_ -> push!(rep_tmp3, tmp2[2]), 1:10)
foreach(_ -> push!(rep_tmp3, tmp2[3]), 1:10)



println("Repeated tmp3:")
println(rep_tmp3)

println("\nRepeated tmp4:")
println(rep_tmp4)
```

```
Repeated tmp3:
Any[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]

Repeated tmp4:
[4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3]
```

Рис. 2.9: Пункт 3.7

10. Далее я создал массив, в котором первый элемент массива tmp встречается 10 раз подряд, второй элемент — 20 раз подряд, третий элемент — 30 раз подряд (2.10).

```
rep_tmp5 = []
rep_tmp6 = [fill(tmp1[1], 10); fill(tmp1[2], 20); fill(tmp1[3], 30)]

foreach(_ -> push!(rep_tmp5, tmp2[1]), 1:10); foreach(_ -> push!(rep_tmp5, tmp2[2]), 1:20)
foreach(_ -> push!(rep_tmp5, tmp2[3]), 1:30)



println("Repeated tmp5:")
println(rep_tmp5)

println("\nRepeated tmp6:")
println(rep_tmp6)
```

```
Repeated tmp5:
Any[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, 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]

Repeated tmp6:
[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]
```

Рис. 2.10: Пункт 3.8

11. Я создал 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(2.11).

```
pow_tmp1 = [fill(2^tmp1[1]); fill(2^tmp1[2]); fill(2^tmp1[3], 4)]
pow_tmp2 = []

push!(pow_tmp2, 2^tmp1[1]); push!(pow_tmp2, 2^tmp1[2])
foreach(_ -> push!(pow_tmp2, 2^tmp1[3]), 1:4)

function six_counter(array)
    amount = 0
    for num in 1:6
        if '6' in string(array[num])
            amount+=1
        end
    end
    return amount
end

println("Powered tmp1:")
println(pow_tmp1)

println("\nPowered tmp2:")
println(pow_tmp2)

println("\nSixes in powered tmp1:")
println(six_counter(pow_tmp1))

println("\nSixes in powered tmp2:")
println(six_counter(pow_tmp2))
```

```
Powered tmp1:
[16, 64, 8, 8, 8, 8]

Powered tmp2:
Any[16, 64, 8, 8, 8, 8]

Sixes in powered tmp1:
2

Sixes in powered tmp2:
2
```

Рис. 2.11: Пункт 3.9

12. Я создал вектор значений $y = e^x cos(x)$ в точках $x = 3, 3.1, 3.2, ..., 6$, поэлементно умножая значения через цикл for и без, после чего с помощью mean нашёл среднее значени у (2.12).

```julia
using Statistics

x = 3:0.1:6
y1 = [exp(i) * cos(i) for i in x]
ymean1 = mean(y1)

y2 = exp.(x) .* cos.(x)
ymean2 = sum(y2) / length(y2)

println("y1 values:")
for i in 1:31
    println(y1[i])
end

println("\ny1 mean values:")
println(ymean1)

println("\ny2 values:")
for i in 1:31
    println(y2[i])
end

println("\ny2 mean values:")
println(ymean2)
```

Рис. 2.12: Пункт 3.10

13. Я создал вектор вида $(x^i, y^j), x = 0.1, i = 3, 6, 9, ..., 36, y = 0.2, j = 1, 4, 7, ..., 34$ через цикл for(2.13).

```julia
x = 0.1
y = 0.2
i_values = 3:3:36
j_values = 1:3:34

res_vec1 = [(x^i, y^j) for i in 3:3:36, j in 1:3:34]
res_vec2 = [(x^i, y^j) for i in i_values, j in j_values]

println("Result vector 1:")
println(res_vec1)

println("\nResult vector 2:")
println(res_vec2)
```

```
Result vector 1:
[(0.0010000000000000002, 0.2) (0.0010000000000000002, 0.0016000000000000003) (0.0010000000000000002, 1.2800000000000006
00000005e-10) (0.0010000000000000002, 6.5536000000000055e-12) (0.0010000000000000002, 5.2428800000000056e-14) (0.001000
(0.0010000000000000002, 2.684354560000004e-20) (0.0010000000000000002, 2.147483648000004e-22) (0.0010000000000000002, 1
0000000000003) (1.0000000000000004e-6, 1.2800000000000006e-5) (1.0000000000000004e-6, 1.0240000000000006e-7) (1.0000000
```

Рис. 2.13: Пункт 3.11

И получил следующие значения для вектора2 (2.14):

```
Result vector 2:
[(0.0010000000000000002, 0.2) (0.0010000000000000002, 0.0016000000000000003) (0.0010000000000000002, 1.2800000000000006e-5) (0.0010000000000000002, 1.0240000000000006e-7) (0.0010000000000000002, 8.192000000000005e-10) (0.0010000000000000002, 6.553600000000055e-12) (0.0010000000000000002, 5.242880000000056e-14) (0.0010000000000000002, 4.194304000000005e-16) (0.0010000000000000002, 3.3554432000000044e-18) (0.0010000000000000002, 2.684354560000004e-20) (0.0010000000000000002, 2.147483648000004e-22) (0.0010000000000000002, 1.7179869184000035e-24); (1.0000000000000004e-6, 0.2) (1.0000000000000004e-6, 0.0016000000000000003) (1.0000000000000004e-6, 1.2800000000000006e-5) (1.0000000000000004e-6, 1.0240000000000006e-7) (1.0000000000000004e-6, 8.192000000000005e-10) (1.0000000000000004e-6, 6.553600000000055e-12) (1.0000000000000004e-6, 5.242880000000056e-14) (1.0000000000000004e-6, 4.194304000000005e-16) (1.0000000000000004e-6, 3.3554432000000044e-18) (1.0000000000000004e-6, 2.684354560000004e-20) (1.0000000000000004e-6, 2.147483648000004e-22) (1.0000000000000004e-6, 1.7179869184000035e-24); (1.0000000000000005e-9, 0.2) (1.0000000000000005e-9, 0.0016000000000000003) (1.0000000000000005e-9, 1.2800000000000006e-5) (1.0000000000000005e-9, 1.0240000000000006e-7) (1.0000000000000005e-9, 8.192000000000005e-10) (1.0000000000000005e-9, 6.553600000000055e-12) (1.0000000000000005e-9, 5.242880000000056e-14) (1.0000000000000005e-9, 4.194304000000005e-16) (1.0000000000000005e-9, 3.3554432000000044e-18) (1.0000000000000005e-9, 2.684354560000004e-20) (1.0000000000000005e-9, 2.147483648000004e-22) (1.0000000000000005e-9, 1.717
```

Рис. 2.14: Пункт 3.11 - вектор 2

14. Я создал вектор с элементами $2^i/i, i = 1, 2, ..., M, M = 25$ используя push! и циклы for(2.15).

```
div_vec1 = []
div_vec2 = [2^i / i for i in 1:25]

M = 25
power = collect(1:M)

for p in power
    div_vec1 = push!(div_vec1, 2^p / p)
end


println("Division vector 1:")
println(div_vec1)

println("\nDivision vector 2:")
println(div_vec2)
```

```
Division vector 1:
Any[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]

Division vector 2:
[2.0, 2.0, 2.6666666666666665, 4.0, 6.4, 10.666666666666666, 18.285714285714285, 32.0, 56.888888888888886, 102.4, 186.1818181818182, 341.3333333333333, 630.1538461538462, 1170.2857142857142, 2184.5333333333333, 4096.0, 7710.117647058823, 14563.555555555555, 27594.105263157893, 52428.8, 99864.38095238095, 190650.18181818182, 364722.0869565217, 699050.6666666666, 1.34217728e6]
```

Рис. 2.15: Пункт 3.12

15. Я создал вектор вида (“fn1”, “fn2”, ..., “fnN”), N=30, через цикл for, добавляя канждому новому элементу fn с помощью знака $ значение i(2.16).

```
N = 30

fn_vec1 = ["fn$i" for i in 1:N]
fn_vec2 = []

for n in N fn_vec2 = push!(fn_vec1, "fn$n") end

println("Fn vector 1:")
println(fn_vec1)

println("\nFn vector 2:")
println(fn_vec2)
```

```
Fn vector 1:
["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", "fn30"]

Fn vector 2:
["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", "fn30"]
```

Рис. 2.16: Пункт 3.13

16. Я создал массив squares, в котором будут храниться квадраты всех целых чисел от 1 до 100, используя цикл, возводя во вторую степень переменную итерации (2.17).

```
size = 100
squares = [i^2 for i in 1:size]
println("Squares array:")
println(squares)
```

```
Squares array:
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361, 400, 441, 484, 529, 576, 625, 676, 729, 784, 841, 900, 961, 1024, 1089, 1156, 1225, 1296, 1369, 1444, 1521, 1600, 1681,
1764, 1849, 1936, 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, 6561, 6724, 6889, 7056, 7225, 7396, 7569, 7744, 7921, 8100, 8281, 8464, 8649, 8836, 9025, 9216, 9409, 9604, 9801, 10000]
```

Рис. 2.17: Пункт 4

17. Подключив библиотеку prime я создаю массив со 169 простыми элементами myprimes, далее я определяю наименьшкий 89 элемент и сохраняю отедльным массивом срез с 89 элемента по 99 (2.18).

```
using Primes

myprimes = primes(1000)[1:168]
least_number= myprimes[89]
prime_arr_cut = myprimes[89:99]

println("Primes Array: ")
println(myprimes)

println("\nLeast 89th prime number: ")
println(least_number)

println("\nSlice of 88-98 element: ")
println(prime_arr_cut)
```

```
Primes Array:
[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, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 21
1, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 4
57, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557, 563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647, 653, 659, 661, 673, 677, 683, 691, 701, 709, 719,
727, 733, 739, 743, 751, 757, 761, 769, 773, 787, 797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881, 883, 887, 907, 911, 919, 929, 937, 941, 947, 953, 967, 971, 977, 983, 991, 997]

Least 89th prime number:
461

Slice of 88-98 element:
[461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523]
```

Рис. 2.18: Пункт 5

18. С помощью цикла for я считаю сумму $i^3 + 4i^2$, суммируя значения. То же самое я делаю и для второго выражения, меняя лишь само выражения. Для 3 я добаляю цилк в цикле для умножения скобок (2.19).

```
M = 100
N = 20

sum_res1 = sum(i^3 + 4i^2 for i in 10:M)
sum_res2 = sum((2^i/i) + (3^i/i^2) for i in 1:M/4)
sum_res3 = 1.0 + sum(prod([(2 * i)/(2 * i + 1) for i in 1:n]) for n in 1:N)

println("Summary result: ", sum_res1)
println("Результат выражения: ", sum_res2)
println("Результат выражения: ", sum_res3)
```

```
Summary result: 26852735
Результат выражения: 2.1291704368143802e9
Результат выражения: 7.170891165651219
```

Рис. 2.19: Пункт 6

3 Код программы

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

P = union(intersect(A,B), intersect(A,B), intersect(A,C), intersect(B,C))
println("P equals: ")
println(P)
```

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

In [2]:
```
Set1 = Set(["Hello", "World", 1, 2, 3])
Set2 = Set([1, 2, 3, 4])
Set_bool1 = Set([true, false, false, true])
Set_bool2 = Set([true, false, false, true])

println("Set1:")
println(Set1)
println("\nSet2:")
println(Set2)

println("\n'setdiff' Operation:")
println(setdiff(Set1, Set2))
println("\n'intersect' Operation:")
println(intersect(Set1, Set2))

println("\n'issetequal' Operation:")
println(issetequal(Set_bool1, Set_bool2))
```

```
Set1:
Set(Any["Hello", 2, "World", 3, 1])

Set2:
Set([4, 2, 3, 1])

'setdiff' Operation:
Set(Any["Hello", "World"])

'intersect' Operation:
Set(Any[2, 3, 1])

'issetequal' Operation:
true
```

In [26]:
```
N = 30

arr1 = collect(1:N)
arr2 = vcat(1:N-1, N)

println("Vector1:")
println(arr1)

println("\nVector1:")
println(arr2)
```

```
Vector1:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30]

Vector1:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30]
```

In [4]:
```
N = 25

arr_reverse1 = collect(N:-1:1)
arr_reverse2 = reverse(collect(1:N))


println("Reversed Vector1:")
println(arr_reverse1)

println("\nReversed Vector2:")
println(arr_reverse2)
```

```
Reversed Vector1:
[25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

Reversed Vector2:
[25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

In [5]:
```
N = 20

arr_half1 = collect(1:N-1)
arr_half2 = collect(N-1:-1:1)
arr_combined1 = vcat(arr_half1, N, arr_half2)
arr_combined2 = [1:N; N-1:-1:1]

println("Combined Vector1:")
println(arr_combined1)

println("\nCombined Vector2:")
println(arr_combined2)
```

```
Combined Vector1:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10,
9, 8, 7, 6, 5, 4, 3, 2, 1]

Combined Vector2:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10,
9, 8, 7, 6, 5, 4, 3, 2, 1]
```

In [6]:
```
tmp1 = [4, 6, 3]
tmp2 = [x for x in [4, 6, 3]]

println("tmp1:")
println(tmp1)

println("\ntmp2:")
println(tmp2)
```

```
tmp1:
[4, 6, 3]

tmp2:
[4, 6, 3]
```

In [7]:
```
tmp_filled1 = []
tmp_filled2 = tmp1[1]

foreach(_ -> push!(tmp_filled1, tmp1[1]), 1:10)
for i in 1:10 tmp_filled2 = vcat(tmp2[1], tmp_filled2) end


println("Filled tmp1:")
println(tmp_filled1)

println("\nFilled tmp2:")
println(tmp_filled2)
```

```
Filled tmp1:
Any[4, 4, 4, 4, 4, 4, 4, 4, 4, 4]

Filled tmp2:
[4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4]
```

In [8]:
```
rep_tmp1 = tmp1
rep_tmp2 = repeat(tmp2, inner=10)

for i in 1:9 rep_tmp1 = vcat(tmp1, rep_tmp1) end
println("Repeated tmp1:")
println(rep_tmp1)

println("\nRepeated tmp2:")
println(rep_tmp2)
```

```
Repeated tmp1:
[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]

Repeated tmp2:
[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]
```

In [9]:
```
rep_tmp3 = []
rep_tmp4 = [fill(tmp1[1], 11); fill(tmp1[2], 10); fill(tmp1[3], 10)]

foreach(_ -> push!(rep_tmp3, tmp2[1]), 1:11); foreach(_ -> push!(rep_tmp3, tmp2[2]), 1:10)
foreach(_ -> push!(rep_tmp3, tmp2[3]), 1:10)



println("Repeated tmp3:")
println(rep_tmp3)
```

```
println("\nRepeated tmp4:")
println(rep_tmp4)
```

```
Repeated tmp3:
Any[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]

Repeated tmp4:
[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]
```

In [10]:
```
rep_tmp5 = []
rep_tmp6 = [fill(tmp1[1], 10); fill(tmp1[2], 20); fill(tmp1[3], 30)]

foreach(_ -> push!(rep_tmp5, tmp2[1]), 1:10); foreach(_ -> push!(rep_tmp5, tmp2[2]), 1:20)
foreach(_ -> push!(rep_tmp5, tmp2[3]), 1:30)



println("Repeated tmp5:")
println(rep_tmp5)

println("\nRepeated tmp6:")
println(rep_tmp6)
```

```
Repeated tmp5:
Any[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, 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, 3]

Repeated tmp6:
[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, 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, 3, 3]
```

In [32]:
```
pow_tmp1 = [fill(2^tmp1[1]); fill(2^tmp1[2]); fill(2^tmp1[3], 4)]
pow_tmp2 = []

push!(pow_tmp2, 2^tmp1[1]); push!(pow_tmp2, 2^tmp1[2])
foreach(_ -> push!(pow_tmp2, 2^tmp1[3]), 1:4)

function six_counter(array)
    amount = 0
    for num in 1:6
        if '6' in string(array[num])
            amount+=1
        end
    end
    return amount
end

println("Powered tmp1:")
println(pow_tmp1)

println("\nPowered tmp2:")
println(pow_tmp2)

println("\nSixes in powered tmp1:")
println(six_counter(pow_tmp1))

println("\nSixes in powered tmp2:")
println(six_counter(pow_tmp2))
```

```
Powered tmp1:
[16, 64, 8, 8, 8, 8]

Powered tmp2:
Any[16, 64, 8, 8, 8, 8]

Sixes in powered tmp1:
2

Sixes in powered tmp2:
2
```

In [12]:
```
using Statistics

x = 3:0.1:6
y1 = [exp(i) * cos(i) for i in x]
ymean1 = mean(y1)

y2 = exp.(x) .* cos.(x)
ymean2 = sum(y2) / length(y2)
```

```
println("y1 values:")
for i in 1:31
    println(y1[i])
end

println("\ny1 mean values:")
println(ymean1)

println("\ny2 values:")
for i in 1:31
    println(y2[i])
end

println("\ny2 mean values:")
println(ymean2)
```

```
y1 values:
-19.884530844146987
-22.178753389342127
-24.490696732801293
-26.77318244299338
-28.969237768093574
-31.011186439374516
-32.819774760338504
-34.30336011037369
-35.35719361853035
-35.86283371230767
-35.68773248011913
-34.68504225166807
-32.693695428321746
-29.538816297262983
-25.032529229039966
-18.975233154958957
-11.157417389647478
-1.3620985182057503
10.632038010191998
25.046704998273004
42.09920106253839
61.99663027669454
84.92906736250268
111.0615860420258
140.5250750527875
173.40577640857734
209.73349424783467
249.46844055885668
292.4867067371223
338.5643778585117
387.36034029093076

y1 mean values:
53.11374594642971

y2 values:
-19.884530844146987
-22.178753389342127
-24.490696732801293
-26.77318244299338
-28.969237768093574
-31.011186439374516
-32.819774760338504
-34.30336011037369
-35.35719361853035
-35.86283371230767
-35.68773248011913
-34.68504225166807
-32.693695428321746
-29.538816297262983
-25.032529229039966
-18.975233154958957
-11.157417389647478
-1.3620985182057503
10.632038010191998
25.046704998273004
42.09920106253839
61.99663027669454
84.92906736250268
111.0615860420258
140.5250750527875
173.40577640857734
209.73349424783467
249.46844055885668
292.4867067371223
338.5643778585117
387.36034029093076

y2 mean values:
53.11374594642971
```

In [13]:

```
x = 0.1
y = 0.2
i_values = 3:3:36
j_values = 1:3:34

res_vec1 = [(x^i, y^j) for i in 3:3:36, j in 1:3:34]
res_vec2 = [(x^i, y^j) for i in i_values, j in j_values]
```

```
println("Result vector 1:")
println(res_vec1)

println("\nResult vector 2:")
println(res_vec2)
```

Result vector 1:
[(0.0010000000000000002, 0.2) (0.0010000000000000002, 0.0016000000000000003) (0.0010000000000000002, 1.2800000000000006e-5) (0.0010000000000000002, 1.0240000000000006e-7) (0.0010000000000000002, 8.192000000000005e-10) (0.0010000000000000002, 6.553600000000055e-12) (0.0010000000000000002, 5.2428800000000056e-14) (0.0010000000000000002, 4.194304000000005e-16) (0.0010000000000000002, 3.3554432000000044e-18) (0.0010000000000000002, 2.684354560000004e-20) (0.0010000000000000002, 2.147483648000004e-22) (0.0010000000000000002, 1.7179869184000035e-24); (1.0000000000000004e-6, 0.2) (1.0000000000000004e-6, 0.0016000000000000003) (1.0000000000000004e-6, 1.2800000000000006e-5) (1.0000000000000004e-6, 1.0240000000000006e-7) (1.0000000000000004e-6, 8.192000000000005e-10) (1.0000000000000004e-6, 6.553600000000055e-12) (1.0000000000000004e-6, 5.2428800000000056e-14) (1.0000000000000004e-6, 4.194304000000005e-16) (1.0000000000000004e-6, 3.3554432000000044e-18) (1.0000000000000004e-6, 2.684354560000004e-20) (1.0000000000000004e-6, 2.147483648000004e-22) (1.0000000000000004e-6, 1.7179869184000035e-24); (1.0000000000000005e-9, 0.2) (1.0000000000000005e-9, 0.0016000000000000003) (1.0000000000000005e-9, 1.2800000000000006e-5) (1.0000000000000005e-9, 1.0240000000000006e-7) (1.0000000000000005e-9, 8.192000000000005e-10) (1.0000000000000005e-9, 6.553600000000055e-12) (1.0000000000000005e-9, 5.2428800000000056e-14) (1.0000000000000005e-9, 4.194304000000005e-16) (1.0000000000000005e-9, 3.3554432000000044e-18) (1.0000000000000005e-9, 2.684354560000004e-20) (1.0000000000000005e-9, 2.147483648000004e-22) (1.0000000000000005e-9, 1.7179869184000035e-24); (1.0000000000000008e-12, 0.2) (1.0000000000000008e-12, 0.0016000000000000003) (1.0000000000000008e-12, 1.2800000000000006e-5) (1.0000000000000008e-12, 1.0240000000000006e-7) (1.0000000000000008e-12, 8.192000000000005e-10) (1.0000000000000008e-12, 6.553600000000055e-12) (1.0000000000000008e-12, 5.2428800000000056e-14) (1.0000000000000008e-12, 4.194304000000005e-16) (1.0000000000000008e-12, 3.3554432000000044e-18) (1.0000000000000008e-12, 2.684354560000004e-20) (1.0000000000000008e-12, 2.147483648000004e-22) (1.0000000000000008e-12, 1.7179869184000035e-24); (1.0000000000000009e-15, 0.2) (1.0000000000000009e-15, 0.0016000000000000003) (1.0000000000000009e-15, 1.2800000000000006e-5) (1.0000000000000009e-15, 1.0240000000000006e-7) (1.0000000000000009e-15, 8.192000000000005e-10) (1.0000000000000009e-15, 6.553600000000055e-12) (1.0000000000000009e-15, 5.2428800000000056e-14) (1.0000000000000009e-15, 4.194304000000005e-16) (1.0000000000000009e-15, 3.3554432000000044e-18) (1.0000000000000009e-15, 2.684354560000004e-20) (1.0000000000000009e-15, 2.147483648000004e-22) (1.0000000000000009e-15, 1.7179869184000035e-24); (1.0000000000000008e-18, 0.2) (1.0000000000000008e-18, 0.0016000000000000003) (1.0000000000000008e-18, 1.2800000000000006e-5) (1.0000000000000008e-18, 1.0240000000000006e-7) (1.0000000000000008e-18, 8.192000000000005e-10) (1.0000000000000008e-18, 6.553600000000055e-12) (1.0000000000000008e-18, 5.2428800000000056e-14) (1.0000000000000008e-18, 4.194304000000005e-16) (1.0000000000000008e-18, 3.3554432000000044e-18) (1.0000000000000008e-18, 2.684354560000004e-20) (1.0000000000000008e-18, 2.147483648000004e-22) (1.0000000000000008e-18, 1.7179869184000035e-24); (1.0000000000000012e-21, 0.2) (1.0000000000000012e-21, 0.0016000000000000003) (1.0000000000000012e-21, 1.2800000000000006e-5) (1.0000000000000012e-21, 1.0240000000000006e-7) (1.0000000000000012e-21, 8.192000000000005e-10) (1.0000000000000012e-21, 6.553600000000055e-12) (1.0000000000000012e-21, 5.2428800000000056e-14) (1.0000000000000012e-21, 4.194304000000005e-16) (1.0000000000000012e-21, 3.3554432000000044e-18) (1.0000000000000012e-21, 2.684354560000004e-20) (1.0000000000000012e-21, 2.147483648000004e-22) (1.0000000000000012e-21, 1.7179869184000035e-24); (1.0000000000000012e-24, 0.2) (1.0000000000000012e-24, 0.0016000000000000003) (1.0000000000000012e-24, 1.2800000000000006e-5) (1.0000000000000012e-24, 1.0240000000000006e-7) (1.0000000000000012e-24, 8.192000000000005e-10) (1.0000000000000012e-24, 6.553600000000055e-12) (1.0000000000000012e-24, 5.2428800000000056e-14) (1.0000000000000012e-24, 4.194304000000005e-16) (1.0000000000000012e-24, 3.3554432000000044e-18) (1.0000000000000012e-24, 2.684354560000004e-20) (1.0000000000000012e-24, 2.147483648000004e-22) (1.0000000000000012e-24, 1.7179869184000035e-24); (1.0000000000000015e-27, 0.2) (1.0000000000000015e-27, 0.0016000000000000003) (1.0000000000000015e-27, 1.2800000000000006e-5) (1.0000000000000015e-27, 1.0240000000000006e-7) (1.0000000000000015e-27, 8.192000000000005e-10) (1.0000000000000015e-27, 6.553600000000055e-12) (1.0000000000000015e-27, 5.2428800000000056e-14) (1.0000000000000015e-27, 4.194304000000005e-16) (1.0000000000000015e-27, 3.3554432000000044e-18) (1.0000000000000015e-27, 2.684354560000004e-20) (1.0000000000000015e-27, 2.147483648000004e-22) (1.0000000000000015e-27, 1.7179869184000035e-24); (1.0000000000000017e-30, 0.2) (1.0000000000000017e-30, 0.0016000000000000003) (1.0000000000000017e-30, 1.2800000000000006e-5) (1.0000000000000017e-30, 1.0240000000000006e-7) (1.0000000000000017e-30, 8.192000000000005e-10) (1.0000000000000017e-30, 6.553600000000055e-12) (1.0000000000000017e-30, 5.2428800000000056e-14) (1.0000000000000017e-30, 4.194304000000005e-16) (1.0000000000000017e-30, 3.3554432000000044e-18) (1.0000000000000017e-30, 2.684354560000004e-20) (1.0000000000000017e-30, 2.147483648000004e-22) (1.0000000000000017e-30, 1.7179869184000035e-24); (1.0000000000000018e-33, 0.2) (1.0000000000000018e-33, 0.0016000000000000003) (1.0000000000000018e-33, 1.2800000000000006e-5) (1.0000000000000018e-33, 1.0240000000000006e-7) (1.0000000000000018e-33, 8.192000000000005e-10) (1.0000000000000018e-33, 6.553600000000055e-12) (1.0000000000000018e-33, 5.2428800000000056e-14) (1.0000000000000018e-33, 4.194304000000005e-16) (1.0000000000000018e-33, 3.3554432000000044e-18) (1.0000000000000018e-33, 2.684354560000004e-20) (1.0000000000000018e-33, 2.147483648000004e-22) (1.0000000000000018e-33, 1.7179869184000035e-24); (1.0000000000000002e-36, 0.2) (1.0000000000000002e-36, 0.0016000000000000003) (1.0000000000000002e-36, 1.2800000000000006e-5) (1.0000000000000002e-36, 1.0240000000000006e-7) (1.0000000000000002e-36, 8.192000000000005e-10) (1.0000000000000002e-36, 6.553600000000055e-12) (1.0000000000000002e-36, 5.2428800000000056e-14) (1.0000000000000002e-36, 4.194304000000005e-16) (1.0000000000000002e-36, 3.3554432000000044e-18) (1.0000000000000002e-36, 2.684354560000004e-20) (1.0000000000000002e-36, 2.147483648000004e-22) (1.0000000000000002e-36, 1.7179869184000035e-24)]

Result vector 2:
[(0.0010000000000000002, 0.2) (0.0010000000000000002, 0.0016000000000000003) (0.0010000000000000002, 1.2800000000000006e-5) (0.0010000000000000002, 1.0240000000000006e-7) (0.0010000000000000002, 8.192000000000005e-10) (0.0010000000000000002, 6.553600000000055e-12) (0.0010000000000000002, 5.2428800000000056e-14) (0.0010000000000000002, 4.194304000000005e-16) (0.0010000000000000002, 3.3554432000000044e-18) (0.0010000000000000002, 2.684354560000004e-20) (0.0010000000000000002, 2.147483648000004e-22) (0.0010000000000000002, 1.7179869184000035e-24); (1.0000000000000004e-6, 0.2) (1.0000000000000004e-6, 0.0016000000000000003) (1.0000000000000004e-6, 1.2800000000000006e-5) (1.0000000000000004e-6, 1.0240000000000006e-7) (1.0000000000000004e-6, 8.192000000000005e-10) (1.0000000000000004e-6, 6.553600000000055e-12) (1.0000000000000004e-6, 5.2428800000000056e-14) (1.0000000000000004e-6, 4.194304000000005e-16) (1.0000000000000004e-6, 3.3554432000000044e-18) (1.0000000000000004e-6, 2.684354560000004e-20) (1.0000000000000004e-6, 2.147483648000004e-22) (1.0000000000000004e-6, 1.7179869184000035e-24); (1.0000000000000005e-9, 0.2) (1.0000000000000005e-9, 0.0016000000000000003) (1.0000000000000005e-9, 1.2800000000000006e-5) (1.0000000000000005e-9, 1.0240000000000006e-7) (1.0000000000000005e-9, 8.192000000000005e-10) (1.0000000000000005e-9, 6.553600000000055e-12) (1.0000000000000005e-9, 5.2428800000000056e-14) (1.0000000000000005e-9, 4.194304000000005e-16) (1.0000000000000005e-9, 3.3554432000000044e-18) (1.0000000000000005e-9, 2.684354560000004e-20) (1.0000000000000005e-9, 2.147483648000004e-22) (1.0000000000000005e-9, 1.7179869184000035e-24); (1.0000000000000008e-12, 0.2) (1.0000000000000008e-12, 0.0016000000000000003) (1.0000000000000008e-12, 1.2800000000000006e-5) (1.0000000000000008e-12, 1.0240000000000006e-7) (1.0000000000000008e-12, 8.192000000000005e-10) (1.0000000000000008e-12, 6.553600000000005

5e-12) (1.0000000000000008e-12, 5.2428800000000056e-14) (1.0000000000000008e-12, 4.194304000000005e-16) (1.0000000000000008e-12, 3.3554432000000044e-18) (1.0000000000000008e-12, 2.684354560000004e-20) (1.0000000000000008e-12, 2.147483648000004e-22) (1.0000000000000008e-12, 1.7179869184000035e-24); (1.0000000000000009e-15, 0.2) (1.0000000000000009e-15, 0.0016000000000000003) (1.0000000000000009e-15, 1.2800000000000006e-5) (1.0000000000000009e-15, 1.0240000000000006e-7) (1.0000000000000009e-15, 8.192000000000005e-10) (1.0000000000000009e-15, 6.5536000000000055e-12) (1.0000000000000009e-15, 5.2428800000000056e-14) (1.0000000000000009e-15, 4.194304000000005e-16) (1.0000000000000009e-15, 3.3554432000000044e-18) (1.0000000000000009e-15, 2.684354560000004e-20) (1.0000000000000009e-15, 2.147483648000004e-22) (1.0000000000000009e-15, 1.7179869184000035e-24); (1.0000000000000008e-18, 0.2) (1.0000000000000008e-18, 0.0016000000000000003) (1.0000000000000008e-18, 1.2800000000000006e-5) (1.0000000000000008e-18, 1.0240000000000006e-7) (1.0000000000000008e-18, 8.192000000000005e-10) (1.0000000000000008e-18, 6.5536000000000055e-12) (1.0000000000000008e-18, 5.2428800000000056e-14) (1.0000000000000008e-18, 4.194304000000005e-16) (1.0000000000000008e-18, 3.3554432000000044e-18) (1.0000000000000008e-18, 2.684354560000004e-20) (1.0000000000000008e-18, 2.147483648000004e-22) (1.0000000000000008e-18, 1.7179869184000035e-24); (1.0000000000000012e-21, 0.2) (1.0000000000000012e-21, 0.0016000000000000003) (1.0000000000000012e-21, 1.2800000000000006e-5) (1.0000000000000012e-21, 1.0240000000000006e-7) (1.0000000000000012e-21, 8.192000000000005e-10) (1.0000000000000012e-21, 6.5536000000000055e-12) (1.0000000000000012e-21, 5.2428800000000056e-14) (1.0000000000000012e-21, 4.194304000000005e-16) (1.0000000000000012e-21, 3.3554432000000044e-18) (1.0000000000000012e-21, 2.684354560000004e-20) (1.0000000000000012e-21, 2.147483648000004e-22) (1.0000000000000012e-21, 1.7179869184000035e-24); (1.0000000000000012e-24, 0.2) (1.0000000000000012e-24, 0.0016000000000000003) (1.0000000000000012e-24, 1.2800000000000006e-5) (1.0000000000000012e-24, 1.0240000000000006e-7) (1.0000000000000012e-24, 8.192000000000005e-10) (1.0000000000000012e-24, 6.5536000000000055e-12) (1.0000000000000012e-24, 5.2428800000000056e-14) (1.0000000000000012e-24, 4.194304000000005e-16) (1.0000000000000012e-24, 3.3554432000000044e-18) (1.0000000000000012e-24, 2.684354560000004e-20) (1.0000000000000012e-24, 2.147483648000004e-22) (1.0000000000000012e-24, 1.7179869184000035e-24); (1.0000000000000015e-27, 0.2) (1.0000000000000015e-27, 0.0016000000000000003) (1.0000000000000015e-27, 1.2800000000000006e-5) (1.0000000000000015e-27, 1.0240000000000006e-7) (1.0000000000000015e-27, 8.192000000000005e-10) (1.0000000000000015e-27, 6.5536000000000055e-12) (1.0000000000000015e-27, 5.2428800000000056e-14) (1.0000000000000015e-27, 4.194304000000005e-16) (1.0000000000000015e-27, 3.3554432000000044e-18) (1.0000000000000015e-27, 2.684354560000004e-20) (1.0000000000000015e-27, 2.147483648000004e-22) (1.0000000000000015e-27, 1.7179869184000035e-24); (1.0000000000000017e-30, 0.2) (1.0000000000000017e-30, 0.0016000000000000003) (1.0000000000000017e-30, 1.2800000000000006e-5) (1.0000000000000017e-30, 1.0240000000000006e-7) (1.0000000000000017e-30, 8.192000000000005e-10) (1.0000000000000017e-30, 6.5536000000000055e-12) (1.0000000000000017e-30, 5.2428800000000056e-14) (1.0000000000000017e-30, 4.194304000000005e-16) (1.0000000000000017e-30, 3.3554432000000044e-18) (1.0000000000000017e-30, 2.684354560000004e-20) (1.0000000000000017e-30, 2.147483648000004e-22) (1.0000000000000017e-30, 1.7179869184000035e-24); (1.0000000000000018e-33, 0.2) (1.0000000000000018e-33, 0.0016000000000000003) (1.0000000000000018e-33, 1.2800000000000006e-5) (1.0000000000000018e-33, 1.0240000000000006e-7) (1.0000000000000018e-33, 8.192000000000005e-10) (1.0000000000000018e-33, 6.5536000000000055e-12) (1.0000000000000018e-33, 5.2428800000000056e-14) (1.0000000000000018e-33, 4.194304000000005e-16) (1.0000000000000018e-33, 3.3554432000000044e-18) (1.0000000000000018e-33, 2.684354560000004e-20) (1.0000000000000018e-33, 2.147483648000004e-22) (1.0000000000000018e-33, 1.7179869184000035e-24); (1.000000000000002e-36, 0.2) (1.000000000000002e-36, 0.0016000000000000003) (1.000000000000002e-36, 1.2800000000000006e-5) (1.000000000000002e-36, 1.0240000000000006e-7) (1.000000000000002e-36, 8.192000000000005e-10) (1.000000000000002e-36, 6.5536000000000055e-12) (1.000000000000002e-36, 5.2428800000000056e-14) (1.000000000000002e-36, 4.194304000000005e-16) (1.000000000000002e-36, 3.3554432000000044e-18) (1.000000000000002e-36, 2.684354560000004e-20) (1.000000000000002e-36, 2.147483648000004e-22) (1.000000000000002e-36, 1.7179869184000035e-24)]

In [14]:
```
div_vec1 = []
div_vec2 = [2^i / i for i in 1:25]

M = 25
power = collect(1:M)

for p in power
    div_vec1 = push!(div_vec1, 2^p / p)
end


println("Division vector 1:")
println(div_vec1)

println("\nDivision vector 2:")
println(div_vec2)
```

```
Division vector 1:
Any[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]

Division vector 2:
[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]
```

In [16]:
```
N = 30

fn_vec1 = ["fn$i" for i in 1:N]
fn_vec2 = []

for n in N fn_vec2 = push!(fn_vec1, "fn$n") end
```

```
println("Fn vector 1:")
println(fn_vec1)

println("\nFn vector 2:")
println(fn_vec2)
```

```
Fn vector 1:
["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", "fn30"]

Fn vector 2:
["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", "fn30"]
```

In [27]:

```
size = 100
squares = [i^2 for i in 1:size]
println("Squares array:")
println(squares)
```

```
Squares array:
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361, 400, 441, 484, 529, 576, 625,
676, 729, 784, 841, 900, 961, 1024, 1089, 1156, 1225, 1296, 1369, 1444, 1521, 1600, 1681, 1764, 1849, 1936, 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, 6561, 6724, 6889,
7056, 7225, 7396, 7569, 7744, 7921, 8100, 8281, 8464, 8649, 8836, 9025, 9216, 9409, 9604, 9801, 10000]
```

In [30]:

```
using Primes

myprimes = primes(1000)[1:168]
least_number= myprimes[89]
prime_arr_cut = myprimes[89:99]

println("Primes Array: ")
println(myprimes)

println("\nLeast 89th prime number: ")
println(least_number)

println("\nSlice of 88-98 element: ")
println(prime_arr_cut)
```

```
Primes Array:
[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, 107, 1
09, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233,
239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 37
3, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503,
509, 521, 523, 541, 547, 557, 563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647, 65
3, 659, 661, 673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773, 787, 797, 809, 811,
821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881, 883, 887, 907, 911, 919, 929, 937, 941, 947, 953, 967, 97
1, 977, 983, 991, 997]

Least 89th prime number:
461

Slice of 88-98 element:
[461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523]
```

In [31]:

```
M = 100
N = 20

sum_res1 = sum(i^3 + 4i^2 for i in 10:M)
sum_res2 = sum((2^i/i) + (3^i/i^2) for i in 1:M/4)
sum_res3 = 1.0 + sum(prod([(2 * i)/(2 * i + 1) for i in 1:n]) for n in 1:N)

println("Summary result: ", sum_res1)
println("Результат выражения: ", sum_res2)
println("Результат выражения: ", sum_res3)
```

```
Summary result: 26852735
Результат выражения: 2.1291704368143802e9
Результат выражения: 7.170891165651219
```

In [ ]:

4 Выводы

В ход выполнения работы я изучил несколько структур данных, реализованных в Julia, а также научился применять их и операции над ними для решения задач.