

Лабораторная работа № 2

Структуры данных

Герра Гарсия Паола Валентина

Содержание

Цель работы	2
Задание	2
Теоретическое введение	2
Выполнение лабораторной работы	2
Выводы	10
Список литературы.....	10

Список иллюстраций

Примеры использования кортежей.....	3
Примеры использования словарей.....	3
Примеры использования множеств	4
Примеры использования массивов.....	4
Примеры использования массивов.....	5
Примеры использования массивов.....	5
Задание №1. Работа с множествами.....	5
Задание №2. Примеры операций над множествами элементов разных типов	6
Задание №3. Работа с массивами.....	6
Задание №3. Работа с массивами.....	6
Задание №3. Работа с массивами.....	6
Задание №3. Работа с массивами.....	6
Задание №3. Работа с массивами.....	6
Задание №3. Работа с векторами.....	7
Задание №3. Работа с векторами.....	7
Задание №3. Работа с векторами.....	7
Задание №3. Работа с векторами.....	8
Задание №3. Работа с векторами.....	8
Задание №3. Работа с векторами.....	9
Задание №3. Работа с векторами.....	9
Задание №4	9
Задание №5. Работа с пакетом Primes	10
Задание №6	10

Цель работы

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

Задание

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

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.

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 (элементы индексируются с единицы).

```

In [1]: # пустой кортеж:
() 

Out[1]: () 

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

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

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

In [5]: # именованный кортеж:
x3 = (a=2, b=1+2) 
Out[5]: (a = 2, b = 3) 

In [6]: # длина кортежа x2:
length(x2) 
Out[6]: 3 

In [7]: # обратиться к элементам кортежа x2:
x2[1], x2[2], x2[3] 
Out[7]: (1, 2.0, "tmp")

```

Примеры использования кортежей

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

Словарь – неупорядоченный набор связанных между собой по ключу данных.

```

In [12]: # создать словарь с именем phonebook:
phonebook = Dict("Иванов И.И." => ("867-5309", "333-5544"), "Бухгалтерия" => "555-2368") 
Out[12]: Dict{String, Any} with 2 entries:
"Бухгалтерия" => "555-2368"
"Иванов И.И." => ("867-5309", "333-5544") 

In [13]: # вывести ключи словаря:
keys(phonebook) 
Out[13]: KeySet for a Dict{String, Any} with 2 entries. Keys:
"Бухгалтерия"
"Иванов И.И." 

In [14]: # вывести значения элементов словаря:
values(phonebook) 
Out[14]: ValueIterator for a Dict{String, Any} with 2 entries. Values:
"555-2368"
("867-5309", "333-5544") 

In [15]: # вывести заданные в словаре пары "ключ - значение":
pairs(phonebook) 
Out[15]: Dict{String, Any} with 2 entries:
"Бухгалтерия" => "555-2368"
"Иванов И.И." => ("867-5309", "333-5544") 

In [16]: # проверка вхождения ключа в словарь:
haskey(phonebook, "Иванов И.И.") 
Out[16]: true 

In [17]: # добавить элемент в словарь:
phonebook["Сидоров П.С."] = "555-3344"

```

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.

```

In [21]: # создать множество из четырёх целочисленных значений:
A = Set([1, 3, 4, 5])
Out[21]: Set(Int64) with 4 elements:
5
4
3
1

In [22]: # создать множество из 11 символьных значений:
B = Set("abracadabra")
Out[22]: Set(Char) with 5 elements:
'a'
'd'
'r'
'b'
'k'

In [24]: # проверка эквивалентности двух множеств:
S1 = Set([1,2]);
S2 = Set([2,1]);
issetequal(S1,S2)
Out[24]: false

In [25]: S3 = Set([1,2,2,3,1,2,3,2,1]);
S4 = Set([2,1,3,1]);
issetequal(S3,S4)
Out[25]: true

In [26]: # объединение множеств:
C=union(S1,S2)

```

Примеры использования множеств

Выполним примеры из раздела про массивы (рис. [-@fig:003]-[-@fig:006]).

Массив — коллекция упорядоченных элементов, размещённая в многомерной сетке.
Векторы и матрицы являются частными случаями массивов.

```

In [32]: # создание пустого массива с абстрактным типом:
empty_array_1 = []
Out[32]: Any[]

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

In [34]: # вектор-столбец:
a = [1, 2, 3]
Out[34]: 3-element Vector{Int64}:
1
2
3

In [35]: # вектор-строка:
b = [1 2 3]
Out[35]: 1x3 Matrix{Int64}:
1 2 3

In [36]: # многомерные массивы (матрицы):
A = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
B = [[1 2 3]; [4 5 6]; [7 8 9]]
Out[36]: 3x3 Matrix{Int64}:
1 2 3
4 5 6
7 8 9

```

Примеры использования массивов

Jupyter Lab2_статистический_анализ Last Checkpoint: ayer a la 1:46 (autosaved)

In [38]: # многомерный массив \$2 \times 3 \times 3\$ (2 строки, 3 столбика) элементов
со значениями, случайно распределенными на интервале [0, 1];
C = rand(2,3);

In [39]: # трёхмерный массив:
D = rand(4, 3, 2)

Out [39]: 4x3x2 Array{Float64, 3}:

[1, 1, 1] =	0.243017	0.3922	0.374472
0.674585	0.409556	0.552351	
0.410355	0.504758	0.167631	
0.937101	0.115161	0.462839	

[1, 1, 2] =

0.915572	0.446506	0.446439
0.533272	0.89519	0.954749
0.297983	0.8597538	0.887333
0.858242	0.177156	0.833028

In [40]: # массив из квадратных корней всех целых чисел от 1 до 10;
roots = [sqrt(i) for i in 1:10]

Out [40]: 10-element Vector{Float64}:
1.0
1.4142135623730951
1.7320508075688772
2.0
2.238067977489979
2.449489742783178
2.6457513116455987
2.8284271247461983
3.0
3.1622776601683795

In [41]: # массив с элементами вида $3 \times x^2$,
где x – нечетное число от 1 до 9 (включительно)
ar_1 = [3*x^2 for i in 1:2:9]

Примеры использования массивов

Jupyter Lab2_статистический_анализ Last Checkpoint: ayer a la 1:46 (autosaved)

In [49]: # транспонирование
b'

Out[49]: 6x2 adjoint(::Matrix{Int64}) with eltype Int64:
1 2
3 4
5 6
7 8
9 10
11 12

In [50]: # транспонирование
c = transpose(b)

Out[50]: 6x2 transpose(::Matrix{Int64}) with eltype Int64:
1 2
3 4
5 6
7 8
9 10
11 12

In [51]: # массив 10x5 целых чисел в диапазоне [10, 20];
ar = rand(10:20, 10, 5)

Out[51]: 10x5 Matrix{Int64}:
10 10 15 19 17
18 19 14 15 13
14 18 16 13 11
18 17 19 16 17
10 18 16 15 13
18 18 10 18 12
19 13 11 14 14
11 18 18 20 14
15 20 19 15 16
19 18 11 16 20

In [52]: # выбор всех значений строки в столбце 2:

Примеры использования массивов

Перейдем к выполнению заданий.

1. Даны множества: $A = \{0, 3, 4, 9\}$, $B = \{1, 3, 4, 7\}$, $C = \{0, 1, 2, 4, 7, 8, 9\}$. Найдем $P = A \cap B \cup A \cap C \cup B \cap C$ (рис. [-@fig:007]).

Задания для самостоятельного выполнения

```
In [61]: A=Set([0, 3, 4, 9])  
B=Set([1, 3, 4, 7])  
C=Set([0, 1, 2, 4, 7, 8, 9])  
#P = A ∩ B ∪ A ∩ C ∪ B ∩ C  
P=union(intersect(A, B), intersect(A, C), intersect(B, C))  
print("P=",P)  
P=Set([0, 4, 7, 9, 3, 1])
```

Задание №1. Работа с множествами

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

```
In [62]: #Conjunto se strings
conjunto1=Set(["a", "b", "c"])

#Conjunto de floats
conjunto2=Set([1.5, 2.5, 3.5])

#Operaciones
union(conjunto1, Set(["a", "c", "d"]))
intersect(conjunto2, Set([2.5, 3.5, 4.5]))
```

Out[62]: Set(Float64) with 2 elements:
3.5
2.5

Задание №2. Примеры операций над множествами элементов разных типов

3. Создадим массивы разными способами, используя циклы (рис. [-@fig:009]-[-@fig:019]):

Задание №3. Работа с массивами

```
In [68]: #3.3 arrays con potencias
arrayB<- vcat(fill(2^tmp[1], 1), fill(2^tmp[2], 1), fill(2^tmp[3], 4),
count_6<- count(x->x == 6, arrayB)
printin("Number 6 appears $count_6 times")
```

Задание №3. Работа с массивами

```
In [77]: #3.4
using Statistics
x_vals = 3:0.1:6
y_vals = exp.(x_vals).*cos.(x_vals)
average_y = mean(y_vals)
println("Average of y: $average_y")
```

Average of y: 53.11374594642971

Задание №3. Работа с массивами

```
In [82]: #3.5
x=0.1
y=0.2
vector1 = [x*i for i in 3:3:36]
vector2 = [y*j for j in 1:3:34]
printin("Firstly 3 vector1 elements: {vector1[1:3]}")
```

Задание №3. Работа с массивами

```
In [83]: #3.6
M= 25
vector3= [(2^i)/i for i in 1:M]
Out[83]: 25-element Vector{Float64}:
2.0
2.0
2.6666666666666665
4.8
6.4
10.666666666666666
18.285714285714285
32.0
56.88888888888886
102.4
188.0
341.3333333333333
630.1538461538462
1178.285714285714285
2184.5333333333333
4096.0
7104.175647058023
14563.555555555555
27594.105263157893
52428.8
99864.38095238095
190650.18181818182
364722.0869565217
699850.66666666666
1.34217728e6
```

Задание №3. Работа с векторами

```
In [84]: #3.7
vector4 = ["fn$i" for i in 1:30]
Out[84]: 30-element Vector{String}:
"fn1"
"fn2"
"fn3"
"fn4"
"fn5"
"fn6"
"fn7"
"fn8"
"fn9"
"fn10"
"fn11"
"fn12"
"fn13"
;
"fn19"
"fn20"
"fn21"
"fn22"
"fn23"
"fn24"
"fn25"
"fn26"
"fn27"
"fn28"
"fn29"
"fn30"
```

Задание №3. Работа с векторами

The screenshot shows a Jupyter Notebook interface with the following details:

- Title:** Jupyter Лаб2_статистический_анализ Last Checkpoint: ayer a las 1:46 (autosaved)
- Kernel:** Julia 1.12
- Cells:**
 - In [93]:** #3.8

```
sum = Base.sum
n = 250
x = rand(0:999, n)
y = rand(0:999, n)
vector5 = [y[i+1] - x[i] for i in 1:n-1]
vector6 = [x[i] + 2*x[i+1] - x[i+2] for i in 1:n-2]
vector7 = [sin(y[i]) / cos(x[i+1]) for i in 1:n-1]
#sum
my_sum = sum(exp(-x[i+1]) / (x[i] + 10) for i in 1:n-1)
println("El valor de la suma es: $my_sum")
```

El valor de la suma es: 0.0001465153887463867
 - In [95]:** #3.9

```
indexes = findall(y .>600)
values_y = y[indexes]
values_x_correspondientes = x[indexes]
```
 - Out[95]:** 103-element Vector{Int64}

```
322
815
187
862
695
467
8
184
313
49
190
544
310
694
682
282
464
```

Задание №3. Работа с векторами

Jupyter Lab2_статистический_анализ Last Checkpoint: ayer a las 1:46 (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Julia 1.12

```
In [96]: media_x = mean(x)
vector8 = [abs(x[i]) - media_x]^0.5 for i in 1:n]
Out[96]: 250-element Vector{Float64}:
 20.37164696336553
 13.341514156946355
 8.48551707322515
 17.12362058165698
 17.748352836175978
 13.115029546287726
 20.73634490453985
 18.000111110768177
 17.46902982959817
 16.03134897962626
 5.8286988878586268
 14.24794720564793
 19.026402707816317
 ...
 21.93180339142274
 7.874261070169167
 12.530123702501983
 21.51752773993173
 17.5782826054853936
 17.492978016552363
 4.24794720564793
 19.026402707816317
 7.348741388836596
 13.783993656076532
 19.67221390692974
 4.1235906683374886
```

```
In [97]: max_y = maximum(y)
cercanos = count(y .>= max_y - 200)
Out[97]: 47
```

```
In [98]: pares = count(x .%2 .==0)
```

Задание №3. Работа с векторами

Jupyter Lab2_статистический_анализ Last Checkpoint: ayer a las 1:46 (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Julia 1.12

```
In [99]: pares = count(x .%2 .==0)
impares = n - pares
Out[99]: 113
```

```
In [100]: multiplos_7 = count(x .%7 .==0)
Out[100]: 37
```

```
In [101]: orden = sortperm(y)
x_ordenado = x[orden]
Out[101]: 250-element Vector{Int64}:
 829
 222
 221
 700
 332
 324
 318
 541
 586
 406
 203
 238
 528
 ...
 694
 849
 657
 202
 184
 120
 597
 215
 735
 881
 862
```

Задание №3. Работа с векторами

```

In [102]: top_10 = sort(x, rev=true)[1:10]
Out[102]: 10-element Vector{Int64}:
 999
 994
 986
 983
 981
 980
 978
 976
 970
 969

In [103]: unicos = unique(x)
Out[103]: 223-element Vector{Int64}:
 915
 322
 572
 203
 815
 672
 70
 824
 187
 757
 466
 703
 862
 :
 317
 366
 100
 906
 189
 981
 562

```

Задание №3. Работа с векторами

```

In [104]: squares = [i^2 for i in 1:100]
Out[104]: 100-element Vector{Int64}:
 1
 4
 9
 16
 25
 36
 49
 64
 81
 100
 121
 144
 169
 7921
 8100
 8281
 8464
 8649
 8836
 9025
 9216
 9409
 9604
 9801
 10000

In [107]: using Primes
Primes.add("Primes")
using Primes

#Primeros 168num primos
myprimes = primes(1000) [1:168] #suficientes primos

```

Задание №3. Работа с векторами

4. Создадим массив squares, в котором будут храниться квадраты всех целых чисел от 1 до 100 (рис. [-@fig:020]).

```

In [108]: #89-esimo primo
primo_89= myprimes[89]
println("89-esimo primo: $primo_89")
89-esimo primo: 461

In [109]: #6.1
suma1 = sum([i^3+4i^2 for i in 10:100])
println("Suma 6.1: $suma1")
Suma 6.1: 26852735

```

Задание №4

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:021]).

```
In [110]: #6.2
M = 25
suma2 = sum([(2^i)/i + (3^i)/(i^2) for i in 1:M])
println("Suma 6.2: $suma2")
Suma 6.2: 2.1291704368143802e9
```

Задание №5. Работа с пакетом Primes

6. Вычислим следующие выражения (рис. [-@fig:022]).

```
In [111]: #6.3
function calcular_serie(terminos)
    resultado = 1.0
    numerador = 2
    denominador = 3

    for i in 1:terminos-1
        termino = numerador/denominador
        resultado += termino
        numerador *= (2*i + 2)
        denominador *= (2*i + 3)
    end
    return resultado
end

suma3 = calcular_serie(5)
println("Suma 6.3 (primeros 5 terminos): $suma3")
Suma 6.3 (primeros 5 terminos): 3.0634920634920633
```

Задание №6

Выводы

В результате выполнения данной лабораторной работы я изучила несколько структур данных, реализованных в Julia, научилась применять их и операции над ними для решения задач.

Список литературы