



Лабораторная работа N°2

по дисциплине Компьютерный практикум по статистическому анализу данных

Ход работы

```
Untitled.ipynb
+
Notebook Julia 1.10.5

[52]: # пустой кортеж:
      ()

[52]: ()

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

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

[54]: # кортеж из целых чисел:
      x1 = (1, 2, 3)
      # кортеж из элементов разных типов:
      x2 = (1, 2.0, "tmp")
      # именованный кортеж:
      x3 = (a=2, b=1+2)
      # с вторым и третьим элементами кортежа x1:
      c = x1[2] + x1[3]

[54]: 5

[ ]:
```

```
Untitled.ipynb
+
Notebook Julia 1.10.5

[56]: # создать словарь с именем phonebook:
      phonebook = Dict{"Иванов И.И." => ("867-5309", "333-5544"), "Бухгалтерия" => "555-2368"}
      # вывести ключи словаря:
      keys(phonebook)
      # вывести значения элементов словаря:
      values(phonebook)
      # проверка вхождения ключа в словарь:
      haskey(phonebook, "Иванов И.И.")

[56]: true

[ ]:
```

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

[59]: 10-element Vector{Float64}:
      1.0
      1.4142135623730951
      1.7320508075688772
      2.0
      2.23606797749979
      2.449489742783178
      2.6457513110645907
      2.8284271247461903
      3.0
      3.1622776601683795

[ ]:
```

Ход работы(1)

```
[61]: #объединение пересечений заданных множеств
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, C), intersect(B, C))
```

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

[]:





Ход работы(2)

```
[62]: set1 = Set([1, 2, "abc"])
      set2 = Set(["abc", 4, 5])

      # Операции над множествами
      union_result = union(set1, set2) # Объединение
      intersect_result = intersect(set1, set2) # Пересечение
      setdiff_result = setdiff(set1, set2) # Разность
```

```
[62]: Set{Any} with 2 elements:
      2
      1
```

[]:

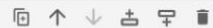


Ход работы(3)

```
[64]: N = 25  
      array1 = collect(20:N)
```

```
[64]: 6-element Vector{Int64}:  
      20  
      21  
      22  
      23  
      24  
      25
```

```
[ ]:
```



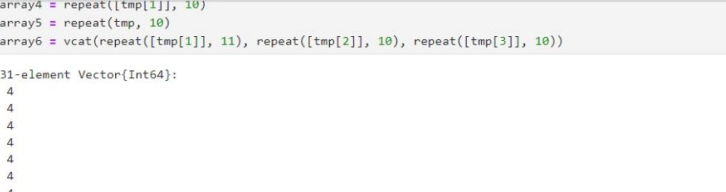


Ход работы(3)

```
[69]: N = 25  
      array1 = collect(21:N)  
      array2 = reverse(array1)
```

```
[69]: 5-element Vector{Int64}:  
      25  
      24  
      23  
      22  
      21
```

```
[ ]:
```



The screenshot shows a Jupyter Notebook window titled "Untitled.ipynb". The top toolbar includes icons for file operations, execution, and code management. The notebook is in "Code" view. The code cell contains three lines of Julia code: `array4 = repeat([tmp[1]], 10)`, `array5 = repeat(tmp, 10)`, and `array6 = vcat(repeat([tmp[1]], 11), repeat([tmp[2]], 10), repeat([tmp[3]], 10))`. The output cell shows the result of the execution: `[74]: 31-element Vector{Int64}:` followed by a list of 31 integers: 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.



Ход работы(3)

```
[77]: N = 25
      array1 = collect(21:N)
      tmp = [4, 6, 3]
      array4 = repeat([tmp[1]], 10)
      array5 = repeat(tmp, 10)
      array6 = vcat(repeat([tmp[1]], 11), repeat([tmp[2]], 10), repeat([tmp[3]], 10))
      array7 = vcat(repeat([tmp[1]], 10), repeat([tmp[2]], 20), repeat([tmp[3]], 30))
      array8 = [2 * tmp[i] for i in 1:3]
      array8 = vcat(array8, repeat([array8[3]], 4))
      count_6 = count(x -> x == 6, array8) # Количество шестерок
      using Statistics
      array9 = [exp(x) * cos(x) for x in 3:0.1:6]
      mean_value = mean(array9)
```

```
[77]: 53 11374594642971
```



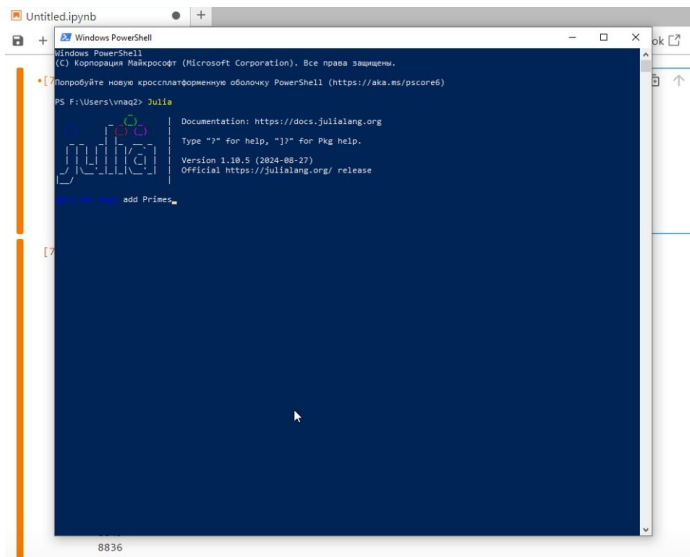

Ход работы(4)

```
[78]: squares = [i^2 for i in 1:100]
```

```
[78]: 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
```

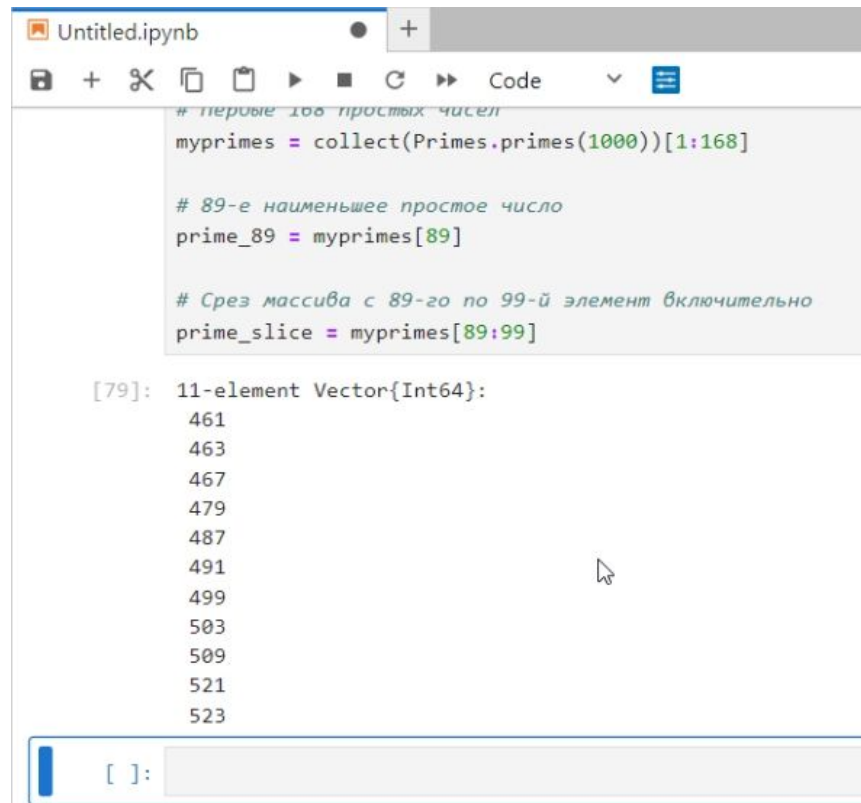
Ход работы(5)



```
Windows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.
[?] Настройте новую кроссплатформенную оболочку PowerShell (https://aka.ms/powershell)
PS F:\Users\vnag2> julia
      _
     _ \
    _ \|
   / _ \|
  / ___ \|
 / /   \|
/_/    _ \|
       / ___ \|
      / /   \|
     / ___ \|
    / ___ \|
   / ___ \|
  / ___ \|
 / ___ \|
/_/    _ \|

Documentation: https://docs.julialang.org
Type "?" for help, "]]" for pkg help.
Version 1.10.5 (2024-08-27)
Official https://julialang.org/ release

julia> julia --help
julia>
```



```
Untitled.ipynb
[+]
[?]
Code
# первые 1000 простых чисел
myprimes = collect(Primes.primes(1000))[1:168]

# 89-е наименьшее простое число
prime_89 = myprimes[89]

# Срез массива с 89-го по 99-й элемент включительно
prime_slice = myprimes[89:99]

[79]: 11-element Vector{Int64}:
 461
 463
 467
 479
 487
 491
 499
 503
 509
 521
 523

[ ]:
```

Ход работы(6)

```
[80]: sum1 = sum(i^3 + 4 * i^2 for i in 10:100)
```

```
[80]: 26852735
```

```
[81]: sum2 = sum(2 * i / (i + 3 * i / i^2) for i in 1:25)
```

```
[81]: 45.79330094261042
```

```
[82]: # Функция для вычисления одного члена ряда с выводом промежуточных значений
function compute_term_debug(n)
    numerator = prod(2:2:2*n) # Числитель: произведение четных чисел
    denominator = prod(3:2:(2*n + 1)) # Знаменатель: произведение нечетных чисел
    term = numerator / denominator
    println("Член $n: $term")
    return term
end

# Сумма первых 10 членов ряда
sum3_debug = sum(compute_term_debug(n) for n in 1:10)
println("Сумма ряда: $sum3_debug")

Член 1: 0.6666666666666666
Член 2: 0.5333333333333333
Член 3: 0.45714285714285713
Член 4: 0.40634920634920635
Член 5: 0.3694083694083694
Член 6: 0.340992340992341
Член 7: 0.31825951825951826
Член 8: 0.29953837012660545
Член 9: 0.2837731927515209
Член 10: 0.27026018357287707
Сумма ряда: 3.945724038603296
```



Вывод

Я изучил несколько структур данных, реализованных в **Julia**, научился применять их и операции над ними для решения задач.